Thèse

pour l'obtention du grade de

Docteur de l'Université de Poitiers

Faculté des sciences fondamentales et appliquées
(Diplôme national – arrêté du 25 avril 2002)

Secteur de recherche : **Informatique**
École Doctorale : **Sciences Pour l'Ingénieur**

présentée par

David FRADIN

Modélisation et simulation d'éclairage à base topologique : application aux environnements architecturaux complexes.

Soutenue le 17 décembre 2004 devant le jury composé de :

René Caubet, Professeur à l'Université de Toulouse ..... Rapporteur
Pascal Guitton, Professeur à l'Université de Bordeaux ..... Rapporteur
Bernard Péroche, Professeur à l'Université de Lyon .....Rapporteur

Michel Mériaux, Professeur à l'Université de Poitiers .....Directeur de Thèse
Daniel Meneveaux, Maître de conférences à l'Université de Poitiers .....Encadrant de thèse
Pascal Lienhardt, Professeur à l'Université de Poitiers ..... Encadrant de thèse

TABLE DES MATIÈRES

CHAPITRE 1

INTRODUCTION

La manipulation et l'utilisation d'environnements architecturaux complexes (les villes, les quartiers, les grands bâtiments meublés) dans les applications informatiques sont en plein essor. Par exemple, les opérateurs téléphoniques utilisent des plans de villes (en 2D ou en 3D) pour positionner les antennes relais et calculer leurs zones de couverture. Les municipalités des grandes villes et l'aménagement du territoire conçoivent leurs grands ouvrages à l'aide de l'informatique. Les visites guidées (vidéos, visites interactives, etc.) et les systèmes de navigation à l'aide du GPS (voiture, téléphone mobile, etc.) demandent des techniques de modélisation et de visualisation de bâtiments ou de villes. Tous ces outils nécessitent des modèles de représentation des données et des algorithmes de visualisation adaptés.

Pour nous donner une idée du nombre de données à manipuler, nous pouvons prendre des exemples réels (figure 1.1). Les tours jumelles du World Trade Center, détruites le 11 septembre 2001, possédaient chacune 110 étages (415 et 416 mètres de haut pour une base carrée de 63 mètres) et un peu plus de $43\ 600$ fenêtres. Elles hébergeaient des bureaux, des hôtels, des restaurants, des magasins, etc. L'Empire State Building est composé de 102 étages agencés en plus de $15\ 000$ pièces. En France, nous pouvons citer la Tour Montparnasse d'une hauteur de 209 mètres et possédant 58 étages. Modéliser de tels bâtiments avec leur mobilier demanderait plusieurs centaines de millions de polygones (une dizaine de meubles par pièce, chacun possédant quelques milliers de faces). À notre connaissance, le record du plus grand nombre de polygones pour un bâtiment est le "UNC Chapel Hill Power Plant", une centrale électrique modélisée avec plus de 12 millions de triangles. Il n'existe que très peu de bâtiments entièrement modélisés. Le plus connu dans la communauté de synthèse d'images est le "Berkeley Soda Hall Building", modélisé par l'équipe de visualisation de Berkeley sous la direction de Sequin et Teller. Ce bâtiment réel possède 7 étages en marche d'escalier (figure 1.2). Son modèle informatique possède environ $1,8$ millions de polygones et est illuminé par $20\ 000$ sources. Ceci donne déjà un grand nombre de polygones à gérer malgré le fait que les meubles soient très simples : sur la figure 1.3 au centre, les chaises sont modélisées à l'aide parallélépipèdes sans arrondis qui manquent de réalisme. Lors de sa thèse à l'IRISA, Meneveaux a aussi modélisé des bâtiments pour des travaux en calcul de radiosité, le plus grand étant visible sur la figure 1.3.

Les bâtiments précédents ont tous été modélisés manuellement. Mais pour des bâtiments très complexes comme un gratte-ciel, une modélisation complète (dessiner les plans, placer les ouver-

Figure 1.1 – *À gauche une photographie de l'Empire State Building, au centre les tours jumelles du World Trade Center, et à droite la Tour Montparnasse.*

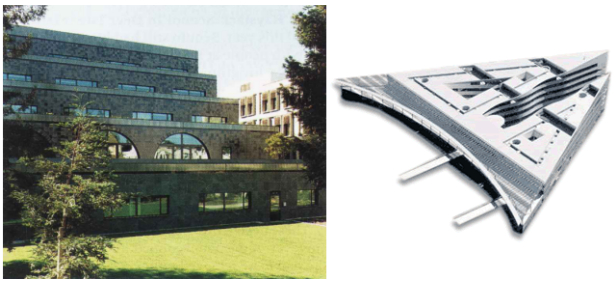


Figure 1.2 – *Photographies du Soda Hall (Berkeley, Californie) et du SP2MI (Futuroscope, France).*

tures, meubler les pièces, ajouter les informations sémantiques) est une tâche ardue qui nécessite plusieurs personnes. Pour qu'un unique utilisateur puisse créer ce type de bâtiment, une génération de sa structure à partir de plans et un ameublement automatique seraient indispensables, mais ce n'est pas le sujet de cette thèse. Nous nous intéressons ici aux structures de données et aux outils permettant la manipulation de ces scènes pour la modélisation et la visualisation. Les exemples étudiés sont donc des bâtiments de taille plus réduite (quelques étages et quelques centaines de pièces) mais possédant déjà un très grand nombre de faces. Considérons par exemple le bâtiment SP2MI (figure 1.2), de l'Université de Poitiers. Bien moins connu que les précédents, il est aussi "plus petit" avec 4 étages et 700 pièces. Pourtant même pour ce bâtiment, l'ameu-

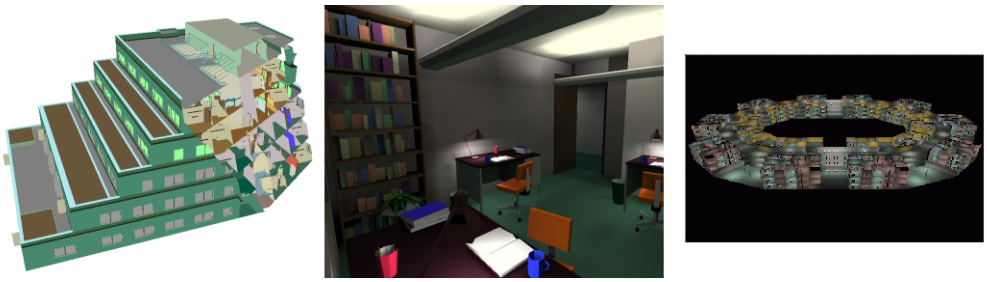


Figure 1.3 – *Images de deux modèles de bâtiments : à gauche le modèle 3D du Berkeley Soda Hall sans meubles et au centre une des pièces meublées, à droite le modèle complet d'un bâtiment issu des travaux de Meneveaux.*

blement s'avère fastidieux. En effet, pour obtenir un maximum de réalisme, l'utilisateur doit placer des meubles et des objets dans toutes les pièces : des tables, des chaises, des fournitures de bureau, etc. De plus, en fonction des meubles utilisés, le nombre de polygones nécessaire à la modélisation d'un bâtiment complet peut atteindre rapidement plusieurs millions.

Le but de cette thèse est de proposer une structure de données, des opérations et des outils qui facilitent la manipulation de grands bâtiments meublés en privilégiant la modélisation de l'intérieur du bâtiment. Ce travail est une première étape vers une future description de villes détaillées. Modéliser et visualiser de manière réaliste une scène architecturale détaillée telle qu'un bâtiment entièrement meublé restent très complexe malgré l'évolution croissante des machines. Premièrement, le nombre de polygones atteint rapidement les limites matérielles fixées par l'architecture de l'ordinateur. Par exemple, si un triangle est uniquement représenté par ses trois sommets, un gigaoctet de mémoire peut contenir jusqu'à 30 millions de triangles. Nous devons néanmoins retirer à cette estimation rapide la taille du système d'exploitation et celle, conséquente, des informations sur les triangles (données photométriques ou sémantiques). De plus, le nombre de primitives est si élevé qu'il devient difficile en des temps acceptables de les modifier voire de les visualiser.

De telles masses de données compliquent les étapes classiques de modélisation et de visualisation. Actuellement, les modeleurs commerciaux (3DS Max, Amapi, ArchiCAD, Lightwave, Maya, etc.) n'ont pas de politique particulière pour la modélisation d'environnements complexes. À notre connaissance, les travaux les plus connus dans le domaine des grands bâtiments n'ont pas encore été mis en place au sein des moteurs de rendus commerciaux (3DS Max, Lightwave, Maya, Mental Ray, etc.) malgré leur intérêt pour les méthodes photoréalistes de calcul de radiosité et de lancer de photons. Si un utilisateur souhaite modéliser de manière détaillée un bâtiment, il doit opérer manuellement. Il décompose lui-même sa scène, les parties sont travaillées indépendamment et sont sauvegardées dans différents fichiers. Une fois la modélisation terminée, l'utilisateur les regroupe pour former le modèle final. Pour la visualisation, le problème est identique ; les parties du bâtiment à afficher sont déterminées manuellement pour simplifier la visualisation et permettre de générer des vidéos de démonstration : suppression des meubles dans

les pièces que l'observateur ne voit pas, utilisation de textures moins détaillées pour les zones éloignées, etc.

Pourtant dans la littérature, de nombreux travaux en synthèse d'images existent sur ce thème. Des modèles topologiques, hiérarchiques, paramétriques ou à niveaux de détail ont été proposés pour modéliser ou visualiser des environnements complexes. Quelques-uns de ces modèles sont présentés dans le chapitre 2. Remarquons néanmoins que ces deux domaines (modélisation et visualisation de complexes architecturaux), bien que concernant la même thématique, ont divergé vers des approches très différentes. En modélisation, le but étant de créer et modifier ces environnements, les études s'intéressent principalement à des modèles les plus génériques possible (formes et structures) et aux opérations de construction et d'édition. La plupart de ces modèles sont assez éloignés des préoccupations d'espace mémoire et de rapidité d'accès aux données. La priorité est donnée à la précision de la description géométrique de l'objet. À notre connaissance, aucune des structures proposées en modélisation n'a encore été implantée dans un modeleur architectural. En rendu réaliste, dans le cas des complexes architecturaux, le problème est tout autre, il s'agit de visualiser un bâtiment le plus rapidement possible et/ou de manière photoréaliste sur les bases des modèles physiques et photogéométriques. Les structures de données proposées ont pour but principal de réduire les temps de calcul liés à la visibilité ; pour les grands bâtiments, des difficultés supplémentaires concernent l'accès aux données et l'encombrement mémoire. Ces environnements ont la particularité de posséder un grand nombre de faces occlusives, c'est-à-dire des polygones cachant une grande partie de la scène. Les structures accélératrices déjà proposées exploitent au maximum cette information. La plupart des techniques ont pour but de trouver les parties de la scène visibles depuis un point donné pour réduire le nombre de primitives à traiter. Pour cela, les polygones sont organisés pour subdiviser la scène de manière efficace. Dans la plupart des cas, l'objectif est retrouver les polygones les plus occlusifs (en général les murs du bâtiment), à créer des groupes de polygones formant des pièces et à déterminer leur organisation dans la scène. Leur but est donc de retrouver des informations topologiques (adjacence, incidence) et sémantiques concernant la scène à partir de simples polygones.

Nous pouvons remarquer que les objectifs des deux domaines sont assez différents malgré un domaine d'application commun. Au fur et à mesure des travaux et de l'augmentation de la puissance des machines, le lien entre la modélisation et la visualisation est devenu de plus en plus ténu. Les informations disponibles dans les structures très complètes de modélisation géométrique ne sont pas exploitées par les algorithmes de visualisation et les besoins en rendu réaliste ne sont pas pris en compte par les modeleurs. L'objectif principal de cette thèse est de proposer une chaîne complète reposant sur une structure de données complète pour les grands bâtiments, pouvant servir depuis la modélisation jusqu'à la visualisation.

Notre premier travail est la définition d'une structure de modélisation géométrique permettant de modéliser de grands bâtiments et prenant en compte les besoins en rendu réaliste : rapidité d'accès aux données, exhaustivité (des murs du bâtiment jusqu'au crayon sur le bureau), informations sémantiques sur la scène (étages, pièces, meubles), informations sur les volumes et les faces (matériaux, données photométriques). De différents travaux en visualisation et en modélisation d'objets complexes, nous avons extrait des points essentiels pouvant être utiles à notre structure :

- les informations topologiques en modélisation simplifient un grand nombre d'opérations géométriques et les relations d'adjacence et d'incidence entre cellules (volumes, faces, etc) sont indispensables à la visualisation ;
- une structuration (partition, hiérarchie, inclusion) permet de travailler localement sur des parties de la scène, de propager des opérations à différents niveaux de précision ;
- une gestion minutieuse des éléments en mémoire offre la possibilité d'avoir l'ensemble des informations nécessaires pour les calculs de modélisation ou de visualisation.

Concernant la structuration du bâtiment, nous souhaitons préciser deux notions devant apparaître au niveau du modèle. D'une part, une structure de *partitions* de la scène doit permettre d'associer des informations sémantiques à des groupes d'objets de l'environnement. Par exemple, nous voulons pouvoir regrouper les pièces en étages ou/et en ailes. À l'intérieur de cette même décomposition, nous souhaitons également réunir les pièces suivant leur utilisation : les bureaux, les salles de cours, etc. La figure 1.4.a montre un exemple de partitions multiples d'un même étage. Ceci peut être utile pour afficher seulement une partie des pièces d'un bâtiment ou pour prévoir des visites guidées en fonction des thématiques de certaines zones (visites de musées). D'autre part, nous souhaitons une description *hiérarchique* du bâtiment pour faciliter son édition en réduisant la quantité d'informations à afficher tout en permettant de travailler localement sur certaines parties de la structure. Par exemple pour meubler une pièce, il est inutile de travailler sur la totalité du bâtiment. Cette décomposition aide également à accélérer les calculs de visualisation en fonction du point de vue. Pour une vue extérieure d'un bâtiment, il est possible d'utiliser une représentation du contour du bâtiment avec des textures ou de réaliser un parcours hiérarchique pour n'afficher que les pièces visibles par les fenêtres. La figure 1.4.b représente la hiérarchie d'un bâtiment simple.

Nous proposons ici un modèle hiérarchique à base topologique dédié à la modélisation et à la visualisation réaliste de complexes architecturaux d'intérieur. Cette structure permet de représenter un environnement de manière cohérente comme une subdivision de l'espace $\mathbb{R}^3$. Notre modèle adjoint deux grands principes à un modèle topologique ordonné : une hiérarchie de détails permettant de travailler à différents niveaux de précision (travail local plus rapide), et une multipartition autorisant le regroupement des entités géométriques en fonction de leur sémantique (pièces adjacentes, murs extérieurs et cloison) ou de leurs propriétés photométriques et physiques (sols de mêmes revêtements, mêmes matériaux de construction).

Nous avons défini également des opérations de construction dédiées au modèle proposé (ajouter un détail, grouper, etc.) et des opérations spécifiques à la modélisation d'environnements architecturaux complexes (créer un mur, ajouter une fenêtre, meubler, etc.). À l'aide de ces opérations, nous avons développé un prototype de modeleur de grands bâtiments montrant la pertinence de notre modèle en terme de qualité de modélisation. Ce logiciel nous a permis de créer plusieurs bâtiments complexes aussi bien du point de vue du nombre de faces que des formes.

À partir de cette structure et des bâtiments construits, nous montrons l'utilité d'enrichir des données usuellement utilisées en visualisation par des informations topologiques sur la scène (disposition des pièces, ouvertures, etc.) et de prendre en compte les contraintes d'accès aux ressources pour la visualisation dès l'étape de modélisation. D'une part, la structure nous permet de diminuer le nombre de primitives à traiter pour permettre un accès plus rapide aux données (une seule pièce chargée au lieu du bâtiment). D'autre part, la structuration de la scène nous permet

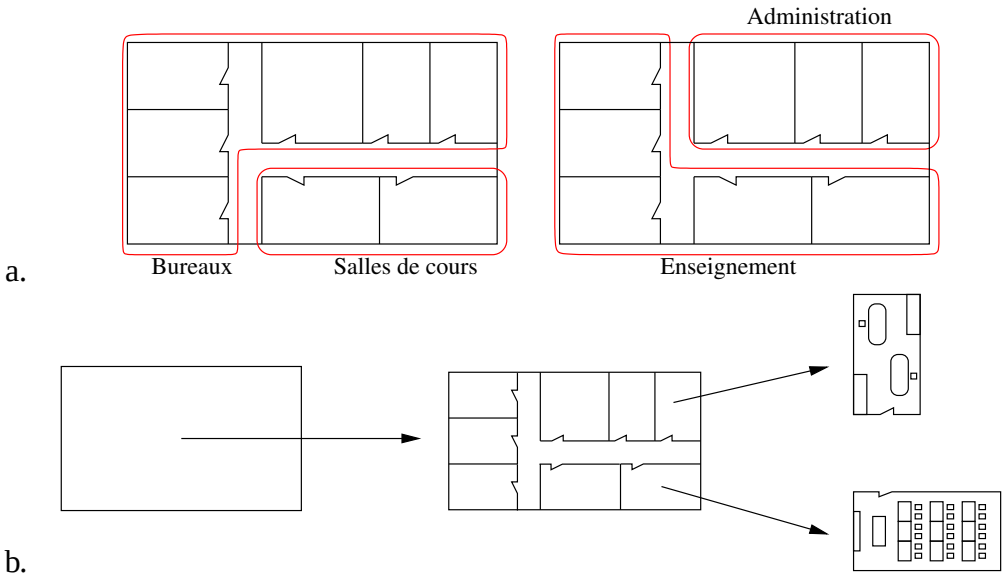


Figure 1.4 – *a. Partition multiple d'un même objet : un étage est subdivisé de deux manières différentes. Chaque subdivision a sa propre sémantique. La première subdivision représente la séparation entre les bureaux et les salles de cours. La seconde met en évidence les parties réservées à l'enseignement et à l'administration. - b. Décomposition d'un objet par niveaux de détail : un étage rectangulaire est détaillé par un ensemble de pièces dont certaines sont meublées (un bureau et une salle de cours).*

une gestion fine de la mémoire grâce aux nombreuses informations disponibles. Nous avons donc développé des algorithmes de lancer de rayons et de lancer de photons sur ces grands bâtiments. Le premier nous permet une visualisation rapide de l'environnement avec prise en compte de l'éclairage direct des sources lumineuses, et le second plus réaliste fournit une simulation des inter-réflexions.

Pour résumer, nos contributions concernent les points suivants :

- la **multipartition**, permettant de présenter plusieurs décompositions possibles d'un même objet ;
- la **hiérarchie**, complémentaire de la structure précédente et permettant un chargement partiel des données pour des traitements localisés ;
- l'**accès aux données** simplifié et accéléré par la structure proposée de manière à pouvoir traiter des objets de grande taille ;
- une **gestion précise de la mémoire** pour manipuler des données qui occupent normalement plus de place que l'espace mémoire disponible ;
- la **visualisation** des environnements modélisés à l'aide d'un algorithme de lancer de rayons utilisant des informations déduites de la structure topologique ;
- la **simulation d'éclairage** pour de grands bâtiments avec un algorithme de lancer de photons.

Ce mémoire est organisé selon le plan suivant. Nous commençons par présenter un état de l'art portant sur la visualisation et la modélisation de scènes architecturales complexes. Nous y

présentons les différentes structures accélératrices utilisées dans les algorithmes de visualisation et de simulation d'éclairage, ainsi que des modèles géométriques à base topologique permettant la manipulation de grands bâtiments. Nous discutons aussi des points qui nous semblent essentiels pour résoudre notre problématique.

Dans le chapitre suivant, nous proposons notre propre modèle permettant de répondre au but fixé : obtenir une chaîne complète de la modélisation à la visualisation de complexes architecturaux. Ce modèle est basé sur les cartes généralisées [Lie94] auxquelles nous avons ajouté les notions de hiérarchie et de multipartition. Nous présentons ensuite notre prototype de modeleur de bâtiments basé sur cette structure. Ce dernier nous permet de générer les scènes nous servant pour les calculs d'illumination.

Enfin, les deux chapitres suivants montrent comment utiliser les informations géométriques, topologiques et sémantiques issues de la modélisation et contenues dans notre modèle pour accélérer les algorithmes de visualisation et de simulation d'éclairage. Nous nous intéressons dans le premier à un algorithme de lancer de rayons et dans le second au lancer de photons. La dernière partie de ce mémoire présente le bilan des travaux réalisés et les perspectives que nous envisageons.

CHAPITRE 2

ÉTAT DE L'ART

Les travaux en synthèse d'images (modélisation et visualisation) sur les complexes architecturaux n'ont en général que peu de liens entre eux. Les travaux en modélisation portent prioritairement sur la définition exhaustive de l'ensemble des propriétés d'un objet complexe. A contrario, les études en rendu réaliste s'intéressent au calcul d'informations géométriques, sémantiques et topologiques à partir de listes de faces, afin d'accélérer les calculs. Les objectifs et les méthodes peuvent donc paraître peu semblables. L'étude de ces structures nous a néanmoins permis de mettre en évidence certains axes de réflexion et certains points essentiels à la modélisation et au rendu réaliste d'environnements architecturaux complexes.

Nous souhaitons ici introduire l'ensemble des modèles et des méthodes étudiées qui ont permis de mettre au point notre modèle. Pour mieux comprendre la problématique liée aux environnements architecturaux complexes, nous exposons dans un premier temps les différents travaux réalisés concernant l'accélération des algorithmes de visualisation et de simulation d'éclairage dans des environnements quelconques et dans des environnements architecturaux. Nous détaillons en particulier les grilles régulières, les arbres BSP et les travaux de Airey, Teller et Meneveaux, qui nous servent de base de travail pour la mise en place d'un lancer de rayons et d'un lancer de photons dans des complexes architecturaux. Dans un second temps, nous étudions différents modèles topologiques permettant de décrire des subdivisions de l'espace, ainsi que des modèles hiérarchiques à base topologique. Cet état de l'art a pour but de montrer les similitudes et les différences entre les structures accélératrices utilisées en visualisation et les modèles géométriques, ainsi que de déterminer les informations nécessaires à la modélisation et à la visualisation d'un grand bâtiment.

2.1 Visualisation d'environnements complexes

Les algorithmes de visualisation et de simulation d'éclairage nécessitent d'estimer les relations de visibilité entre l'observateur et une face, ou entre une face et une autre face. Les techniques à base de tampon de profondeur (en anglais, *Z-buffer*) ont été les premières à être développées. Elles consistent à projeter sur un écran l'ensemble des faces que voit un observateur en prenant soin de respecter leur ordre de profondeur. Ces opérations forment maintenant les ins-

tructions de base de tous les processeurs de cartes graphiques. La puissance de ces processeurs ne cessant d'augmenter, les techniques de projection sont maintenant très utilisées. Elles servent à réaliser des calculs d'ombrages (*soft shadows* [HLHS03]), de visibilité [HA00] ou d'illumination locale ou globale à l'aide de programmation des fonctions du processeur graphique (*pixel/vertex shaders* [NPG04]).

D'autres techniques à base de tracé de rayons ont aussi été développées [App68]. Le tracé de rayons consiste à calculer le plus vite possible les intersections entre un rayon (une demi-droite) et les faces de la scène. De nombreuses méthodes de calcul d'éclairage ont été développées à partir du tracé de rayons (en anglais *ray casting*). Le lancer de rayons (*ray tracing*) [Whi80] et le tracé de chemin lumineux (*path tracing*) [Kaj86] simulent le trajet inverse de la lumière (de l'œil à la source), alors que le lancer de photons [JC95] et le calcul de radiosité [GTGB84] ont pour objectif de simuler les inter-réflexions diffuses et/ou spéculaires en propageant l'énergie lumineuse dans la scène depuis les sources lumineuses.

Pour la plupart de ces méthodes, les scènes sont simplifiées afin d'accélérer le temps de calcul. Les objets sont décrits à l'aide des polygones ayant un minimum de sommets, généralement des triangles en raison de leur simplicité d'affichage, aussi bien pour les algorithmes de *Z-buffer* (algorithmes de remplissage de polygones) que pour ceux de tracé de rayons (optimisation de l'intersection rayon/triangle). Les environnements sont donc très souvent définis à l'aide d'une simple liste de faces, et l'utilisation d'une structure accélératrice permet d'accélérer les temps de calculs.

Nous commençons par décrire les structures généralement utilisées pour accélérer les algorithmes de visualisation. Puis nous détaillons les méthodes dédiées à la visualisation et à la simulation d'éclairage dans de grands bâtiments.

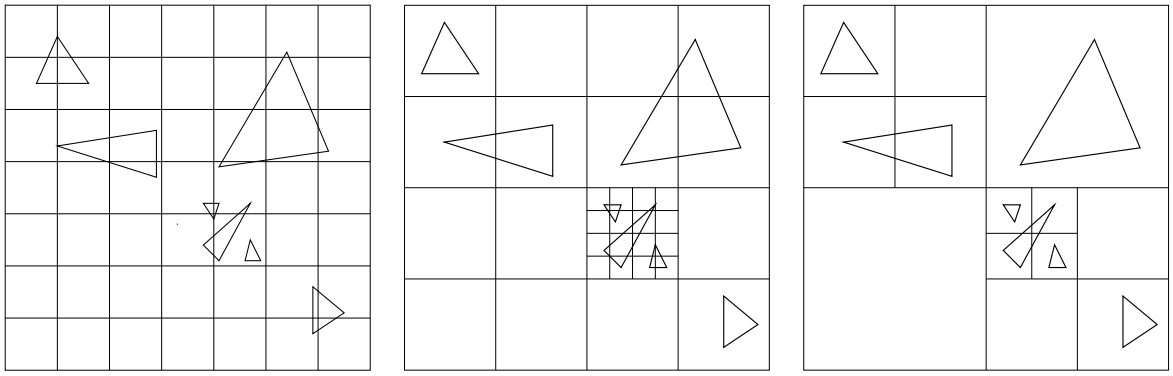
2.1.1 Structures accélératrices classiques

Le principal intérêt de ces méthodes est de réduire le nombre de polygones à traiter afin de diminuer les temps de calcul. Une approche de type "diviser pour mieux régner" est donc en général utilisée pour diminuer le nombre d'objets à manipuler : réduire le nombre de tests d'intersections dans un tracé de rayons, ou le nombre de projections dans un tampon de profondeur. Nous présentons ici des structures accélératrices fréquemment utilisées en visualisation.

Subdivisions spatiales

Ces techniques de subdivision spatiale consistent à appliquer un schéma de découpage pouvant s'adapter à la scène. Elles sont très utilisées pour accélérer les algorithmes à base de tracé de rayons (détaillé dans le chapitre 5). L'idée est de ne pas tester pour un rayon donné l'ensemble des faces lors de la recherche de l'intersection la plus proche. Pour cela, la scène est subdivisée en sous-parties et seuls les polygones appartenant aux parties traversées par le rayon sont testés. Le nombre de polygones contenus dans une partie étant en général très petit devant le nombre total, l'accélération du calcul peut être très importante. Nous présentons ici 4 techniques de subdivision : la grille régulière, la multigrille, l'arbre octal et l'arbre BSP.

Les méthodes basées sur les grilles régulières [FTI00] consistent à subdiviser l'espace de manière uniforme (figure 2.1.a). À chaque voxel est associée la liste des faces qu'il contient. Les volumes sont tous identiques et subdivisent l'espace de la scène sans prendre en compte les objets eux-mêmes. Pour construire ces grilles, des algorithmes de discrétisation sont utilisés pour déterminer quels voxels sont touchés par une face. Pour lancer un rayon dans une grille, l'algorithme de tracé de droite de Bresenham en 3D permet de parcourir la grille très rapidement. C'est cette rapidité de parcours qui a fait de la grille régulière une structure très utilisée en tracé de rayons.



a. b. c.

Figure 2.1 – *Trois différentes méthodes de découpage de l'espace : a. la grille régulière. - b. la multigrille. - c. le quadtree.*

La grille régulière a pour inconvénient de ne pas tenir compte du déséquilibre de la scène en termes de disposition des faces dans l'espace. Si la scène manque de régularité, de nombreux voxels parcourus peuvent être vides et certains peuvent contenir trop de faces. Afin de mieux s'adapter à la scène, il est possible de construire une *multigrille*, ou une hiérarchie de grilles uniformes [CDP95] (figure 2.1.b). Si un voxel de la grille possède un trop grand nombre de faces pour rapport aux autres, il est à nouveau subdivisé par une autre grille. Ceci permet, lorsque des scènes sont irrégulières, de conserver une rapidité de parcours des voxels comme pour une grille régulière.

En 1984, Glassner [Gla84] propose d'utiliser une subdivision spatiale non uniforme : l'arbre octal (en anglais *octree*). La discrétisation de l'espace se fait alors en régions cubiques de tailles variables. La construction se fait de manière descendante. Au départ l'arbre ne possède qu'un nœud qui représente le cube englobant la scène. Puis de manière récursive, si un nœud de l'arbre possède beaucoup de faces, il est subdivisé en 8 régions égales. Nous obtenons alors une scène découpée en fonction de la répartition des polygones dans l'espace. La figure 2.1.c montre un exemple d'octree. Les scènes irrégulières sont donc beaucoup mieux représentées à l'aide de cette structure d'arbre, mais le coût de parcours d'un octree est plus important que celui d'une grille.

Une autre technique permettant de diminuer le nombre de primitives géométriques à traiter est la subdivision binaire (*Binary Space Partitioning* ou BSP). Proposée par Fuchs en 1980

[FKN80], elle consiste à découper une scène en deux sous-scènes selon un plan de découpage. Chaque sous-scène obtenue est à son tour subdivisée selon un nouveau plan. Chaque noeud correspond à un plan de coupe et possède un fils droit et un fils gauche donnant les faces respectivement devant et derrière le plan. Initialement, Fuchs propose de choisir le plan de coupe de manière à avoir un arbre équilibré pour une recherche ultérieure plus rapide. Cet équilibre s'obtient en choisissant toujours un plan permettant de séparer les polygones en deux groupes équitables. L'arbre est construit récursivement jusqu'à avoir classé l'ensemble des faces (figure 2.2). Les faces intersectées par un plan de coupe apparaissent dans les deux sous-arbres à la fois. D'autres choix du plan de coupe ont aussi été proposés : prendre le polygone le plus occlusif ou minimiser le nombre de polygones intersectés par le plan. Le choix du meilleur plan de coupe est un problème difficile, car il dépend énormément de la configuration de la scène.

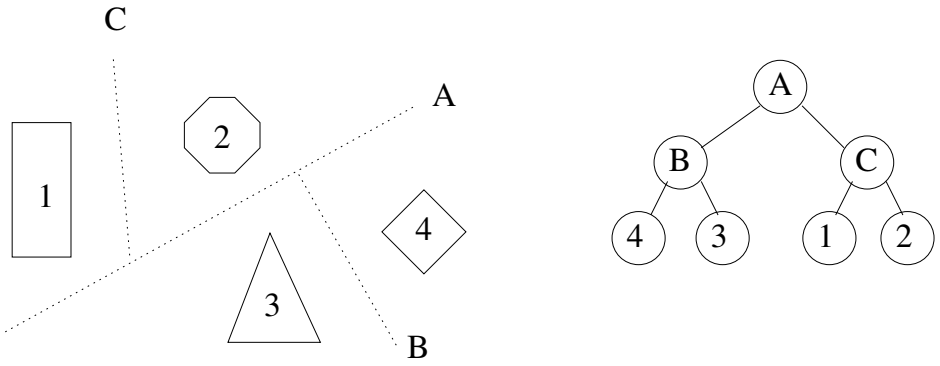


Figure 2.2 – *Exemple d'un arbre BSP en 2D : à droite la scène et trois plans de coupes, à gauche l'arbre BSP correspondant.*

Si les plans sont choisis perpendiculairement aux axes du repère, l'arbre BSP particulier est aussi appelé un arbre kD (ou *kD-tree*). Une utilisation très efficace des arbres kD a été réalisée par Wald [Wal04] pour le développement d'un lancer de rayons interactif, nommé RTRT.

Volumes englobants

Afin d'accélérer les calculs d'intersection entre un rayon et la scène, une autre méthode consiste à regrouper les faces entre elles et à créer des volumes autour de ces groupes. L'intérêt est encore de réduire le nombre de polygones testés. Les intersections sont alors calculées avec les groupes (moins nombreux) puis avec les faces du groupe au premier plan.

Kay et Kajiya proposent en 1986 une technique de détermination de volumes englobants convexes [KK86]. La construction de ces volumes se réalise à partir d'un ensemble de paires de plans parallèles. La figure 2.3 montre à droite un exemple de volume englobant créé à partir de paires de plans parallèles. Lorsque le volume englobant de chaque objet a été créé, une hiérarchie est construite à l'aide d'une technique de découpage médian (*median cut*). Chaque liste de faces est coupée en deux sous-listes, elles-mêmes subdivisées ensuite de manière récursive. Un arbre binaire est en même temps construit et décrit la scène de manière hiérarchique. Cette technique

permet d'accélérer les tests d'intersections entre un rayon et les groupes de la scène. Si un rayon ne rencontre pas le volume englobant d'un nœud de l'arbre, alors il ne rencontre pas non plus l'ensemble de ses fils. Ainsi, seul un petit nombre des faces de la scène entre en compte dans le calcul d'intersection.

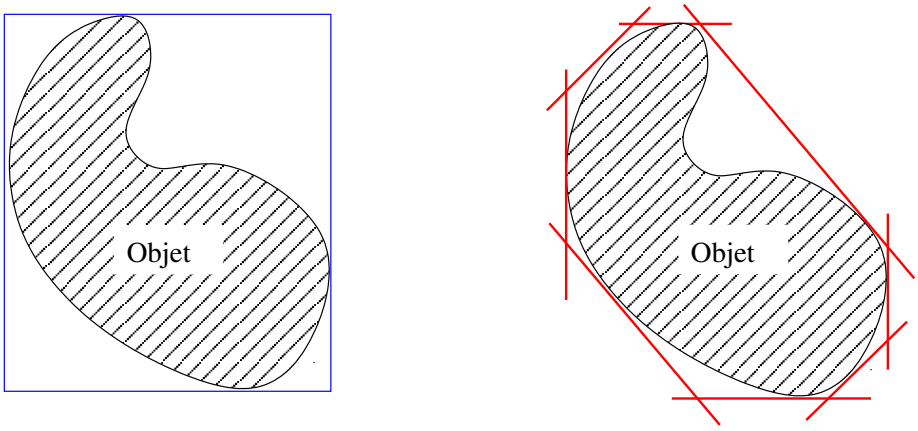


Figure 2.3 – *À gauche, un objet délimité par sa boîte englobante. - À droite, le même objet est enfermé dans des bounding slabs : technique de découpage de scène consistant à définir des paires de plans parallèles autour de l'objet pour obtenir des limites plus précises qu'avec les boîtes englobantes.*

Goldsmith et Salmon [GS87] proposent un procédé un peu différent. Leur but n'est pas d'obtenir le volume englobant s'adaptant le mieux à la forme de l'objet mais de calculer les intersections entre le rayon et la hiérarchie de volumes le plus rapidement possible. La hiérarchie de boîtes englobantes (des parallélépipèdes rectangles) se construit en minimisant de façon heuristique le nombre moyen d'intersections avec des rayons aléatoires. La hiérarchie est un arbre binaire créé de manière ascendante. Les feuilles de l'arbre représentent les boîtes englobantes élémentaires (des groupes de faces dans un volume englobant) et les nœuds forment de nouveaux groupes à partir des précédents. La géométrie de la scène se situe aux feuilles de l'arbre, les nœuds internes ne contenant que des coordonnées de boîtes englobantes.

Niveaux de détails

Les niveaux de détails (en anglais Levels Of Detail, LOD) permettent d'utiliser plusieurs descriptions d'un même objet en fonction de la distance qui les sépare du point de vue. En effet, pour un objet lointain, il n'est pas nécessaire d'afficher tous les détails. Cette technique permet de réduire le nombre de faces totales dans une scène complexe et donc d'en accélérer l'affichage. Dans une scène, seuls les objets importants possèdent des niveaux de détail. La modélisation de ces différents niveaux est longue et demande une grande précision. De nombreux algorithmes performants de simplification de maillage ont été développés [HG97, Lue01, SGG+00]. Cette technique est très efficace pour accélérer les affichages en *Z-Buffer*. Pour cela, le maillage affiché

est choisi en fonction de la distance observateur/objet et d'une estimation de l'erreur d'affichage entre deux niveaux. Néanmoins, le choix du bon niveau de détail en fonction de la distance à l'observateur ainsi que la transition entre deux niveaux sont des points difficiles à implanter.

Certaines techniques proposées peuvent produire aussi des discontinuités du maillage qui rendent le modèle erroné. Des informations topologiques permettent de corriger ces erreurs [PH97, DPM97]. Remarquons aussi que les niveaux de détails sont peu utilisés dans des calculs d'éclairement global. En effet, les tests d'intersection sont parfois faussés, car il est fréquent que le maillage d'un niveau simplifié n'inclue pas au sens géométrique l'ensemble des faces d'un niveau plus détaillé. De plus, le fait de choisir un niveau de détail particulier en fonction de la distance complique considérablement le calcul des ombres et des inter-réflexions. Par exemple, l'ombre d'un objet lointain risque d'être obtenue à l'aide de sa simplification, ce qui ôte du réalisme si l'ombre est proche de l'observateur. Les niveaux de détails ne permettent donc pas de calculer de manière précise les échanges lumineux s'opérant dans un bâtiment. Les LOD permettent néanmoins de diminuer les temps de calcul de radiosité [DB00]. Ils sont tout de même plus souvent utilisés avec des techniques à base de *Z-Buffer* comme dans [MPB03].

2.1.2 Les environnements architecturaux et urbains complexes

Nous détaillons ici les modèles fréquemment utilisés en visualisation d'environnements architecturaux. Si le lecteur souhaite plus d'informations sur les différentes techniques existantes, il peut se référer à l'état de l'art de Cohen-Or [COCSD03] sur les problèmes de visibilité. La plupart des modèles prennent en compte les particularités des scènes architecturales. Les bâtiments sont composés de pièces et n'ont pas une disposition régulière des faces : certaines pièces sont très meublées et d'autres non, certains étages n'ont pas forcément la même forme que les autres, etc. Enfin, ces scènes possèdent un grand nombre de polygones occlusifs, c'est-à-dire des polygones cachant une grande partie de la scène. Par exemple, le sol, les murs et le plafond d'une pièce cachent toutes les faces des étages inférieurs et supérieurs. L'objectif est donc de détecter ces polygones, dits occlusifs (*occluders*), et de les utiliser pour subdiviser la scène afin d'accélérer les calculs.

2.1.3 Les villes

Nous ne souhaitons pas décrire ici l'ensemble des techniques permettant d'accélérer la visualisation de scènes urbaines complexes, puisque ceci sort du cadre de cette thèse. Nous en citons néanmoins quelques unes proposées dans ce domaine. L'objectif est de montrer que la problématique est similaire à celle des grands bâtiments : utiliser les particularités de la scène afin de trouver les polygones occlusifs.

La modélisation en $2D\frac{1}{2}$ est dédiée à la description simplifiée d'environnements urbains pour des calculs efficaces de visibilité. Le principe est de décrire le bâtiment sous forme de polygones dans un plan horizontal et d'attribuer une hauteur à chacun. De nombreuses techniques ont été conçues avec ce modèle pour rendre interactives des visites de grandes villes (*walkthrough*). Elles proposent de précalculer des zones de visibilité au sein de la scène urbaine. Wonka [WWS00] propose de voir les environnements urbains comme des scènes en 2,5D puisque les calculs de

visibilité entre subdivisions spatiales 3D est un problème complexe. La figure 2.4 illustre le gain en nombre de polygones grâce à la méthode. Les polygones réellement chargés en mémoire sont mis en évidence et les bâtiments non utilisés sont représentés par leur silhouette au sol. Des méthodes proposées par Schaufler [SDDS00] et Durand [DDTP00] utilisent des modèles 3D simplifiés. Ces algorithmes peuvent manipuler une plus grande variété de scènes en comparaison de la $2D\frac{1}{2}$, mais sont beaucoup plus coûteux en temps de calcul à cause de l'augmentation de la complexité des scènes. Toutes ces techniques reposent sur le fait que les façades des bâtiments masquent une grande partie de la scène. Par exemple, un piéton dans une ville ne voit en général qu'une très petite partie de cette ville à cause des bâtiments cachant la vue.

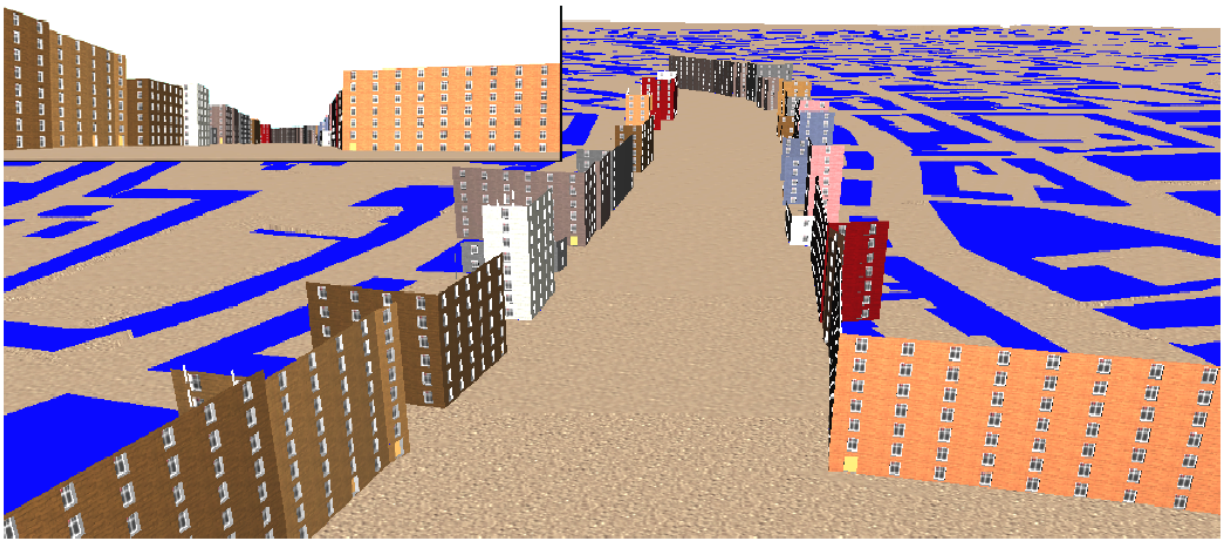


Figure 2.4 – *L'image vue par l'observateur figure en haut à gauche, et le reste de l'image montre une vue du dessus de la ville, afin de mettre en évidence le peu de façades chargées en mémoire (images issues de l'article de Wonka).*

Les plus occlusifs sont supposés être verticaux et reliés à la terre (les façades de bâtiments). Néanmoins, décrire un environnement urbain avec uniquement des polygones extrudés verticalement et agrémentés de textures n'est pas suffisant pour notre problématique. Par exemple, l'intérieur des bâtiments ne peut pas être meublé de cette façon. Lors de notre réflexion sur un modèle, nous garderons à l'esprit qu'avoir une description suffisamment simple de l'extérieur du bâtiment peut être intéressant, afin d'utiliser plus tard ces techniques pour des vues d'extérieur, voire des visites de villes composées de nos bâtiments.

2.1.4 Les grands bâtiments

Le principe de précalcul de visibilité est de connaître l'ensemble des polygones qui sont visibles depuis un point (observateur, lumière, élément de surface). Lorsque le nombre de polygones est très grand (cas des bâtiments), ce prétraitement est primordial afin de réduire les temps

de calcul pour la simulation d'éclairage et/ou la visualisation. En général, calculer un ensemble plus grand de polygones potentiellement visibles (*potentially visible set PVS*) est plus simple qu'obtenir une solution exacte au problème. La recherche des PVS dans une scène complexe se fait généralement de deux manières différentes :

- en 2D, en projetant les objets ou les boîtes englobantes dans un espace de travail avant de déterminer leur visibilité. Ici les calculs se font souvent de manière interactive à l'aide de la carte graphique et pour une visibilité œil/scène, et sont très utilisés par des visites de bâtiment à l'aide de *Z-Buffer* [GKM93, LG95] ;
- en 3D, avec des calculs de visibilité à l'intérieur du maillage de la scène [AB90, TS91, DDP97, COFHZ98, MBSB03]. Il s'agit alors d'un précalcul long indépendant du point de vue et dans tout le bâtiment. Ces techniques sont très utilisés pour des simulations d'éclairage telles que le calcul de radiosité.

La méthode la plus utilisée pour les bâtiments en 3D est l'arbre BSP. En 1990, Airey [AB90] est le premier à s'intéresser concrètement aux complexes architecturaux. Il utilise le principe des ouvertures (en anglais *portals*) pour accélérer la visualisation en environnement architectural [AB90]. Les plans de coupe choisis sont les polygones occlusifs de la scène (murs, plafonds, etc.). Le processus récursif de subdivision se termine quand il ne reste dans chaque sous-scène qu'un petit nombre de ces polygones. Les feuilles de l'arbre binaire ainsi obtenues représentent des petites parties de la scène appelées cellules (en anglais *cells*). Pour faciliter les calculs, Airey considère que les polygones occlusifs sont alignés avec les axes du repère. Il utilise donc un arbre kD pour découper ses scènes, et les boîtes alignées avec les axes ainsi obtenues sont appelées cellules.

Teller a étendu cette méthode BSP à des environnements complexes quelconques [TS91]. Les murs n'étant plus axiaux, le plan de découpage choisi est le polygone le plus occlusif. La recherche de ce polygone est complexe et difficile à mettre en œuvre et cette technique demande à l'époque plusieurs heures de précalcul pour de grands bâtiments. Les feuilles de l'arbre BSP correspondent à des cellules convexes et les ouvertures sont automatiquement déduites à l'aide d'opérations booléennes. Ces informations sont ensuite utilisées pour estimer les relations de visibilité entre les cellules. Il propose aussi une solution analytique au problème de visibilité entre deux ouvertures. La figure 2.5 illustre le résultat du calcul de visibilité entre une pièce et les autres pièces et entre un observateur et le bâtiment. Les zones mises en évidence ne représentent qu'une petite partie du bâtiment, ceci permet de réduire grandement le nombre de primitives à prendre en compte et d'accélérer les calculs de visualisation.

Luebke en 1995 [LG95] utilise le principe de cellules reliées par des ouvertures pour réaliser un programme de visite interactive (*walkthrough*) de bâtiments. Pour cela, il propose une adaptation de l'algorithme de visualisation en tampon de profondeur (*Z-Buffer*). Il suppose connue la décomposition du bâtiment en cellules et en ouvertures. Il signale que ce découpage peut s'obtenir à l'aide des algorithmes précédents ou d'un modeleur pour générer une partition correspondant mieux à la scène (cellules non convexes). Afin de visualiser la scène, il projette sur le plan image à l'aide de la carte graphique la pièce dans laquelle l'observateur se situe. Pour chaque ouverture (et chaque miroir) visible depuis ce point de vue, il détermine dans le plan de l'image une boîte englobante 2D de l'ouverture et renvoie à la carte graphique la pièce correspondante avec un fenêtrage (*clipping*) adapté à cette boîte. Une fois toutes les ouvertures traitées,

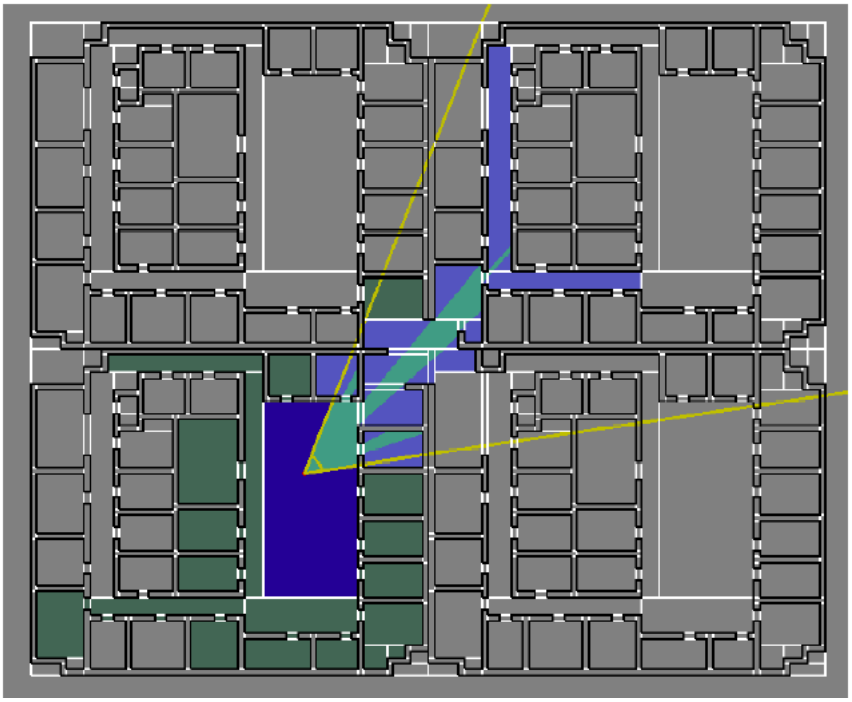


Figure 2.5 – *Image montrant qu'à partir d'un point de vue donné seul un petit nombre de pièces du bâtiment est réellement visible (issue d'un article de Teller).*

une image complète est créée. Cela revient à calculer dynamiquement les PVS depuis un point de vue donné. Les tests ont été réalisés sur des maillages de radiosité contenant $367\ 000$ triangles.

Meneveaux [MBMD98] propose de retrouver de manière automatique ou semi-automatique les pièces des bâtiments selon une méthode à base de règles. Une première étape consiste à regrouper des polygones quasi coplanaires (correspondant aux murs du bâtiment). Pour cela, les polygones sont représentés par des points dans un espace dual : coordonnées polaires des normales des polygones dans le plan xOy (espace de Hough). En regroupant ces points par proximité, Meneveaux retrouve les plans les plus occlusifs qui servent de plans de découpage. Il définit ensuite un ensemble de règles dédiées au découpage d'environnements d'intérieur pour déduire de ces plans : des cellules (les plus proches possible des pièces), leurs ouvertures et leur graphe d'adjacence. Enfin, un graphe de visibilité est calculé à partir de ces données. Ce graphe correspond aux PVS de Airey et Teller.

Dans [MBSB03], Meneveaux propose également de réduire la taille des PVS en affinant la taille des cellules : elles ne correspondent plus aux pièces, mais à de plus petits groupes de polygones (*clusters*) tels que des meubles ou des parties de meubles. Les cellules étant plus fines, le calcul de visibilité est plus long, mais les PVS obtenues sont plus précises.

2.1.5 Discussion

L'extraction d'informations à partir de la liste des faces représentant une scène complexe nécessite beaucoup de calculs. Ceux-ci sont coûteux à la fois en espace mémoire, puisque tous les polygones résident en mémoire, et en temps de calcul, car de très nombreuses données doivent être traitées. De plus, la mise en oeuvre peut être très complexe. Par exemple, Airey et Teller utilisent des heuristiques de subdivision choisies empiriquement ; le choix du polygone comme nœud de l'arbre BSP se fait soit suivant le polygone médian pour équilibrer l'arbre BSP, le polygone le plus occlusif pour éliminer un maximum de faces ou encore afin de réduire le nombre de polygones coupés par le plan. Meneveaux utilise un processus plus coûteux à base de règles qui elles aussi supposent les murs verticaux et les pièces convexes. Enfin, ces méthodes sont sensibles aux erreurs numériques (sommets non alignés, faces dégénérées, faces très petites).

Les techniques de rendu réaliste spécialisées en environnements architecturaux ne se soucient pas de la manière dont ils ont été modélisés ou estiment que certaines informations sont perdues. L'objectif est de calculer des informations topologiques (en général incomplètes) alors que l'étape de modélisation peut fournir toutes les informations géométriques, topologiques et sémantiques nécessaires si ces dernières sont stockées. Remarquons aussi que la reconstruction d'informations topologiques à partir d'un simple maillage (problème similaire à la recherche de cellules dans une liste de faces) est un problème ne trouvant pas de solution générale.

Les informations sémantiques sont aussi très importantes. Par exemple, les faces des meubles sont petites et très proches, donc elles n'ont pas grand intérêt à être présentes lors des calculs de découpage. Seules les faces des murs, sols et plafonds sont nécessaires pour calculer un découpage. Mais savoir si une face appartient à un mur ou à un meuble est une information sémantique que seul le modeleur peut fournir. Une simple liste de faces ne permettant pas cette distinction, il est possible d'utiliser plusieurs listes : une pour les murs, et une pour les meubles de chaque pièce ou de chaque étage (c'est le cas du modèle du Soda Hall). Mais cette décomposition sommaire de la scène est alors faite manuellement ou de manière semi-automatique. Pourtant, elle pourrait être déduite aisément de la phase de modélisation, sans erreur de calcul ni heuristique, en utilisant simplement des informations topologiques et sémantiques.

2.2 Modélisation géométrique à base topologique

De nombreux modeleurs ont été conçus et développés pour une utilisation industrielle :

- archiCAD/autoCAD de l'entreprise Autodesk (architecture, mécanique, cartographie, etc.) ;
- CATIA de Dassault Systemes (automobile, aérospatial, électronique, etc.) ;
- 3DS max de Discreet ou Maya d'Alias (effets spéciaux, jeux vidéo, design).

La précision numérique est l'une des principales préoccupations. Par exemple, une pièce mécanique peut être construite au micron près. Les opérations de manipulation doivent être définies de manière à conserver un maximum de précision et à réduire les erreurs numériques, quelles que soient les caractéristiques géométriques de l'objet modélisé : ses dimensions (une vis ou un moteur d'avion), son nombre de sommets (maillage simple ou complexe), etc.

Afin de mieux comprendre comment les objets complexes sont représentés et manipulés en modélisation géométrique, nous précisons ici certaines méthodes et structures de données permettant de conserver un grand nombre d'informations géométriques, sémantiques et topologiques. Notre objectif est de faire ressortir parmi l'ensemble des informations pouvant être représentées celles qui seront pertinentes en visualisation. Nous montrons l'importance des informations topologiques durant la modélisation et justifions leur utilisation dans notre modèle. Nous verrons dans les chapitres suivants comment optimiser les algorithmes de visualisation et les calculs de visibilité à l'aide de certaines de ces informations.

2.2.1 Modélisation géométrique

Classiquement, les objets géométriques considérés sont des ensembles de points de $\mathbb{R}^n$, généralement $\mathbb{R}^3$. Si le nombre de points formant l'objet est infini, il est évident qu'une description explicite est impossible. Une représentation implicite est donc nécessaire. De nombreuses méthodes de modélisation géométriques ont été proposées depuis plus de 30 ans. Par exemple, beaucoup de travaux portent sur la définition et la manipulation de courbes et surfaces implicites, paramétriques, etc. Une méthode classique de construction d'objets $3D$ à partir de primitives $2D$ est la méthode dite de balayage (en anglais *sweeping*) [PK86]. L'objet $3D$ est engendré par le déplacement d'une forme géométrique $2D$ (appelé générateur) le long d'une trajectoire. Une extension de cette méthode permet de créer des *cylindres généralisés* [AB76] pour lesquels le générateur peut changer de taille, d'orientation et de forme. L'extrusion et les cylindres de révolution (figure 2.7) sont des cas particuliers de ces modèles.

Deux approches classiques permettant de représenter et de manipuler des objets définis comme des assemblages d'objets géométriques élémentaires sont la *géométrie constructive de solides* (*constructive solid geometry*, CSG) [RV82] et la représentation par les bords (*boundary representation*, B-rep).

Le principe de la CSG est de définir un objet géométrique comme le résultat de l'application d'opérations booléennes régularisées (union, intersection, différence) à des primitives géométriques. Initialement, des primitives simples étaient utilisées (des sphères, des parallélépipèdes, etc.). Actuellement, les primitives sont plus élaborées, comme des demi-espaces définis par des surfaces libres. L'objet est représenté par un arbre CSG, décrivant les primitives et les opérations de construction utilisées (figure 2.6).

La modélisation à base topologique, issue de la représentation par les bords[1], a pour objectif de modéliser des objets géométriques subdivisés, i.e. partitionnés en cellules de différentes dimensions (0 pour les sommets, 1 pour les arêtes, 2 pour les faces, 3 pour les volumes, etc.). Les cellules présentent entre elles des relations d'incidence et d'adjacence. Le principe de cette modélisation est de représenter la topologie (structure) de la subdivision par un modèle combinatoire (par exemple la structure d'arêtes ailées [Bau72]), auquel sont associées des informations décrivant la forme de l'objet (modèle de plongement), par exemple les points associés aux sommets, les courbes associées aux arêtes, la couleur aux faces, etc.

[1] *Le principe initial de la représentation par les bords est de représenter un solide (i.e. un volume de $\mathbb{R}^3$) par la surface décrivant son bord. Traditionnellement, une subdivision de cette surface est en fait représentée.*

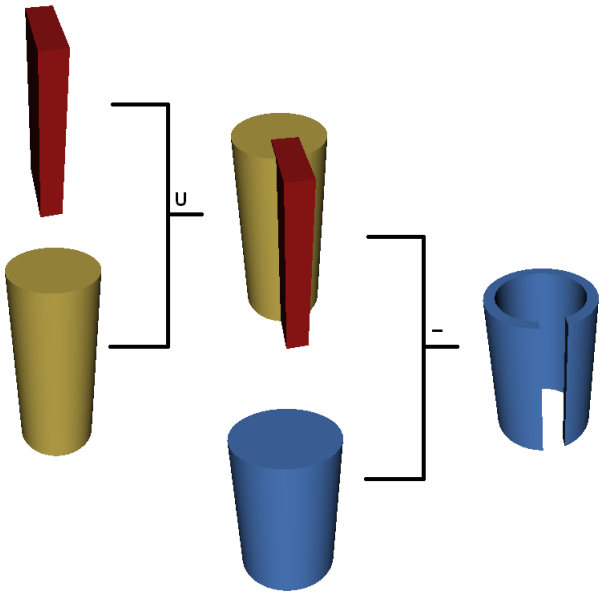
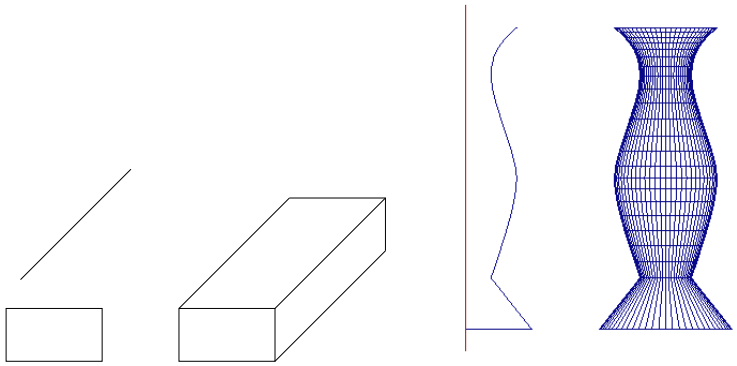


Figure 2.6 – *Exemple d'arbre CSG composé d'une union et d'une différence.*



a. b.

Figure 2.7 – *a. Un exemple d'extrusion, un rectangle se déplace le long d'un segment pour donner un parallélépipède. - b. Un exemple de révolution, un profil tourne autour d'un axe pour donner un vase.*

2.2.2 Modélisation à base topologique

De nombreux travaux ont été menés afin de définir des structures combinatoires permettant de représenter la topologie d'objets subdivisés (par exemple [Bau72], [Wei85], [Bri89], [Lie89]).

Les informations topologiques structurent les informations géométriques et permettent une manipulation efficace de ces dernières. Lors d'une transformation géométrique, des erreurs d'approximation numérique peuvent intervenir et deux surfaces confondues avant transformation

peuvent devenir disjointes. Comme la relation d'adjacence entre deux faces est explicite, les erreurs de calcul peuvent ainsi être détectées, voire même évitées. La topologie est considérée comme prépondérante dans ces modèles par rapport aux informations de forme.

De nombreux modèles topologiques ont été proposés afin de représenter différentes classes d'objets géométriques. Historiquement, les premiers travaux ont porté sur la représentation de subdivision de surfaces (variétés de dimension 2), puis sur les complexes (non-variétés) de dimension 2. Il existe à l'heure actuelle une grande diversité de modèles permettant de représenter de manière efficace des objets de dimension quelconque, depuis des objets très généraux (complexes de dimension n) jusqu'à des sous-classes de ces derniers (variétés, variétés orientables sans bord, quasi-variétés, etc.).

Modèle simplicial et modèle cellulaire

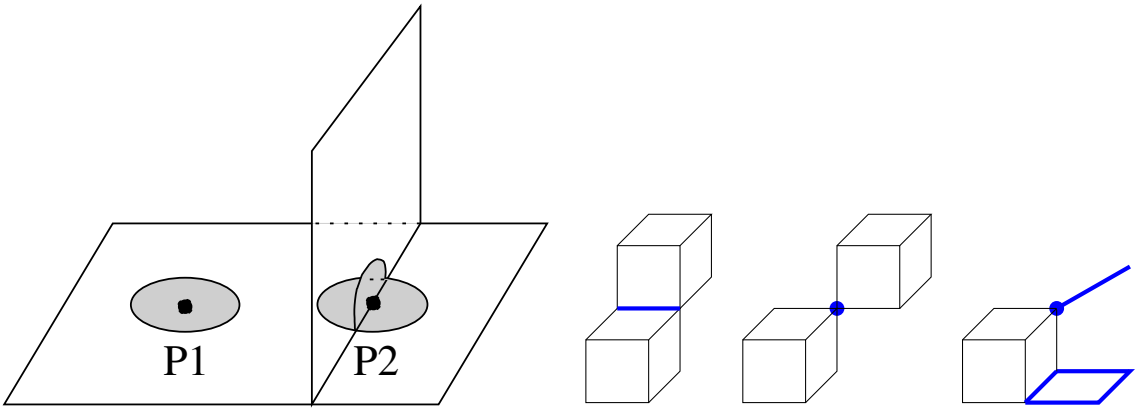
Ces différents modèles peuvent aussi être distingués suivant le type de cellules composant l'objet. Les modèles simpliciaux permettent de représenter des objets composés de simplexes (sommets, arêtes, faces triangulaires, volumes tétraédriques, etc.). Pour d'autres modèles, les cellules sont régulières (cubiques, simploïdales, etc.). Ces modèles sont particulièrement intéressants pour la représentation de maillages : la subdivision n'est pas liée à une sémantique particulière et certains calculs complexes sont simplifiés par la régularité de la structure des cellules (par exemple, l'intersection rayon/triangle).

Les modèles cellulaires permettent de représenter des subdivisions d'objets en cellules quelconques. Une partie très importante des travaux en modélisation géométrique porte sur la définition de ces modèles, car de nombreuses applications (CAO, modélisation géologique, etc.) nécessitent d'associer une sémantique (concrètement des informations telles qu'une terminologie) aux cellules de l'objet modélisé. Par exemple, le volume d'une couche géologique peut avoir une structure quelconque. De même, une face d'une pièce mécanique n'est pas forcément triangulaire ni cubique.

Complexes et variétés

Les objets réels sont tous des variétés de dimension 3, c'est-à-dire constitués de volumes. Par exemple, un ensemble de couches géologiques est composé de volumes qui partagent des faces ; même une feuille de papier est un volume très fin. Plus formellement, une variété (en anglais *manifold*) est un objet où le voisinage de tout point est homéomorphe à une boule (voir figure 2.8.a). Une variété subdivisée de dimension n est un assemblage de cellules de dimension n le long de cellules de dimension $n-1$. En dimension 2, les faces sont donc assemblées le long de leurs arêtes et deux faces au plus partagent une même arête.

Un complexe (*non-variétés*, en anglais *non-manifolds*) est un objet plus général ne répondant pas au critère précédent. Il peut être construit par assemblage de cellules de dimension quelconque le long de cellules de dimensions inférieures (et non de dimension $n-1$ seulement). La figure 2.8.b montre des exemples de complexes. En pratique, ce type d'objets n'est utile que lors de calculs particuliers (extraction de squelettes, etc.) pour la représentation de simplification d'objets. Par exemple, une antenne vue de loin peut être assimilée à une arête.



a. b.

Figure 2.8 – *a. Le voisinage du point P1 est homéomorphe à un disque (boule de dimension 2), mais le voisinage du point P2 ne l'est pas, donc l'objet n'est pas une variété. - b. Exemples de complexes en dimension 3 : en gras, les cellules dont le voisinage n'est pas homéomorphe à une sphère.*

Les graphes d'incidence et les modèles ordonnés

Les graphes d'incidence représentent les différentes cellules d'une subdivision par les nœuds d'un graphe, et les relations de bord entre cellules (relations d'incidence) par les arêtes de ce graphe. Ils peuvent être définis en dimension quelconque [Ede87, Sob89]. Il existe aussi d'autres types de graphes basés sur la représentation explicite d'adjacence entre cellules (e.g. graphes d'adjacence [AFF85]). Ces structures sont aussi utilisées en analyse d'images (graphes d'adjacence de régions ou RAG, etc.).

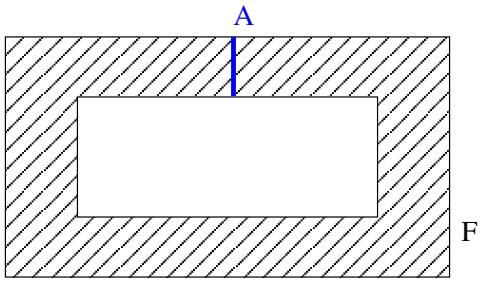


Figure 2.9 – *La multiincidence est le fait qu'une cellule de la subdivision en plusieurs fois incidente à une autre cellule. Ici la face F est incidente deux fois à l'arête A.*

Les principaux problèmes liés à la définition même des graphes d'incidence et d'adjacence sont les suivants :

– ils ne permettent pas de représenter sans ambiguïté tous types d'objets, en particulier la multiincidence (figure 2.9) ;

– les relations d'ordre entre cellules (en $2D$ par exemple, l'ordre des arêtes autour d'un sommet) ne sont pas explicitement représentées ; or cette information d'ordre permet d'optimiser de nombreux algorithmes ;
– il est très difficile d'exprimer les contraintes de cohérence que doit vérifier un graphe d'incidence afin de représenter une classe particulière de subdivisions (par exemple pour ce qui nous concerne, les variétés de dimension 3 [Wei88]).

Ces différents problèmes expliquent pourquoi les modèles ordonnés ont été introduits. Ils ont pour avantage principal d'être définis de manière algébrique à l'aide d'un unique type d'élément de base sur lequel des applications sont définies. Différents modèles ont été proposés en fonction des classes de subdivision à modéliser (dimension, orientabilité, etc.). Lienhardt [Lie91] a montré que tous ces modèles sont équivalents au domaine de modélisation près.

Les modèles ordonnés représentent, comme leur nom l'indique, une relation d'ordre au sein des cellules incidentes. Ceci facilite par exemple la détection de l'orientabilité de l'objet modélisé. Les entités de base varient en fonction des modèles. Un des premiers modèles à avoir été proposé est basé sur les arêtes : les arêtes ailées (en anglais, *winged edges*) [Bau72, Wei85]. Ce modèle est principalement utilisé pour la modélisation de subdivisions de surfaces orientables. Le modèle des *cartes combinatoires* [Edm60] se base sur des entités plus élémentaires appelées *brin*, correspondant intuitivement en $2D$ à des demi-arêtes. Ceci permet de représenter des subdivisions de surfaces orientables sans bord. Elles ont été étendues par Lienhardt [Lie89] en dimension quelconque pour la modélisation de variétés orientables ou non, avec ou sans bord (voir section 3.1).

Une structure équivalente, les structures *cell-tuples*, a été définie en parallèle par Brisson [Bri89]. Des extensions de ces structures ont été proposées pour la représentation de complexes cellulaires en toute dimension [Elt94].

2.2.3 Modèles hiérarchiques

En raison du grand nombre d'objets à manipuler dans une scène complexe, de nombreux travaux ont été réalisés pour proposer des structures reposant sur les structures topologiques précédentes et intégrant une notion de hiérarchie pour permettre de manipuler la scène à différents niveaux de détail. Ils sont communément appelés dans la littérature *modèles hiérarchiques* ou *modèles multirésolutions* (par exemple [Kro95], [DF88], [CDM$^+$94], [PFP95]). Ils permettent de représenter l'objet à différents niveaux de complexité géométrique, et de répartir les traitements à effectuer sur les différents niveaux selon les besoins.

Kropatsch *et al.* [Kro95] ont défini les pyramides d'images dans le cadre de l'analyse d'images. Celles-ci représentent l'image sous forme d'une hiérarchie de partitions et permettent ainsi d'appliquer les algorithmes de traitements sur des petites parties de l'image. La partition initiale, appelée *base* de la pyramide, correspond à l'ensemble des pixels. Les niveaux suivants permettent d'agréger les pixels en régions puis les régions entre elles. Ici, les relations topologiques essentielles sont les relations de connexité entre pixels. Ainsi, les premiers niveaux de la pyramide permettent de décomposer une image en régions homogènes, en utilisant uniquement des critères simples d'homogénéité (couleurs ou textures semblables), alors que des fusions se situant à un plus haut niveau dans la pyramide permettent d'assembler ces régions en objets. L'aspect hié-

rarchique des pyramides permet donc de mémoriser détail par détail les composants d'un objet. Par exemple, l'image d'une table à un certain niveau de la pyramide est issue de la fusion de plusieurs régions codant les différents éléments de celle-ci (pieds, plateau). Cette technique permet de transformer, en une suite de traitements locaux, des algorithmes travaillant antérieurement sur l'ensemble d'une image pour ainsi réduire les temps de calcul ou avoir des résultats étape par étape grâce à une analyse multiéchelle.

De Floriani *et al.* proposent un modèle à base topologique appelé *Multiresolution Simplicial Model* [DPM97]. Celui-ci tente de synthétiser la plupart des structures de multirésolution (LOD, pyramides, etc.) en formalisant le concept de description d'un maillage à plusieurs niveaux de détails en toutes dimensions à l'aide d'un graphe orienté acyclique de complexes simpliciaux. Le graphe est construit de la manière suivante : chaque nœud correspond à une *modification* s'appliquant à une partie du maillage de l'objet, et chaque arête représente le lien de dépendance entre les différentes modifications. De Floriani définit une modification comme étant une succession d'opérations de base sur les simplexes. Cette structure a été généralisée à des complexes cellulaires : *multiresolution meshes* [DM01]. Pour permettre de définir les modifications comme précédemment, De Floriani fixe des opérations de base sur les cellules. En $2D$, ces opérations sont les suivantes : subdivision d'un arbre quaternaire ou *quadtree* (un carré est remplacé par quatre carrés), triangulation d'un triangle par 4 triangles, division d'un triangle en deux, insertion ou suppression d'un sommet et enfin contracter une arête en un sommet. La figure 2.10 montre un exemple en $2D$ d'un ensemble de modifications appliqué à un polygone, ainsi que l'objet final obtenu après application de toutes les opérations. Pour connaître le résultat d'un ensemble de modifications, une phase d'évaluation doit être effectuée : en fonction des modifications choisies, l'objet obtenu est plus ou moins détaillé. De Floriani remarque que pour certaines modifications, les cellules résultant de l'évaluation risquent de ne plus être conformes à la définition des variétés. Par exemple, si un sommet est inséré sur une arête lors d'une modification d'une face $F1$ et que cette arête est partagée par deux faces $F1$ et $F2$, rien ne garantit que le sommet ait aussi été inséré sur la face $F2$. Elle propose donc une notion de groupes (appelés *clusters*) permettant de regrouper les cellules sur lesquelles une modification influe et d'appliquer cette modification à chacune des cellules concernées pour conserver la cohérence des opérations. Par exemple sur la figure 2.10, les polygones étiquetés 1 et 2 disparaissent pour laisser place à 4 triangles suite à une suppression d'arêtes et à une triangulation. Ici c'est la suppression de l'arête qui oblige à former le groupe. Elle appelle alors *graphe conforme* un graphe similaire au précédent, mais dont toutes les modifications ont lieu sur des groupes et non sur des cellules. La phase d'évaluation et la phase de correction du graphe de construction sont les raisons pour lesquelles nous n'avons pas utilisé le modèle de De Floriani. En effet, il est nécessaire de manipuler une représentation évaluée pour minimiser les coûts de traitement. De plus, ce modèle est défini en dimension quelconque, mais les opérations élémentaires et les modifications n'ont été définies qu'en dimension 2, or nous souhaitons représenter des bâtiments à l'aide de subdivisions de $\mathbb{R}^3$. Néanmoins, les idées suivantes nous paraissent intéressantes à intégrer dans le modèle que nous proposons : le principe de hiérarchisation pour les opérations de modifications, le fait de pouvoir les appliquer sur des parties localisées d'un maillage, et enfin la possibilité de décomposer de plusieurs façons différentes un même groupe de polygones.

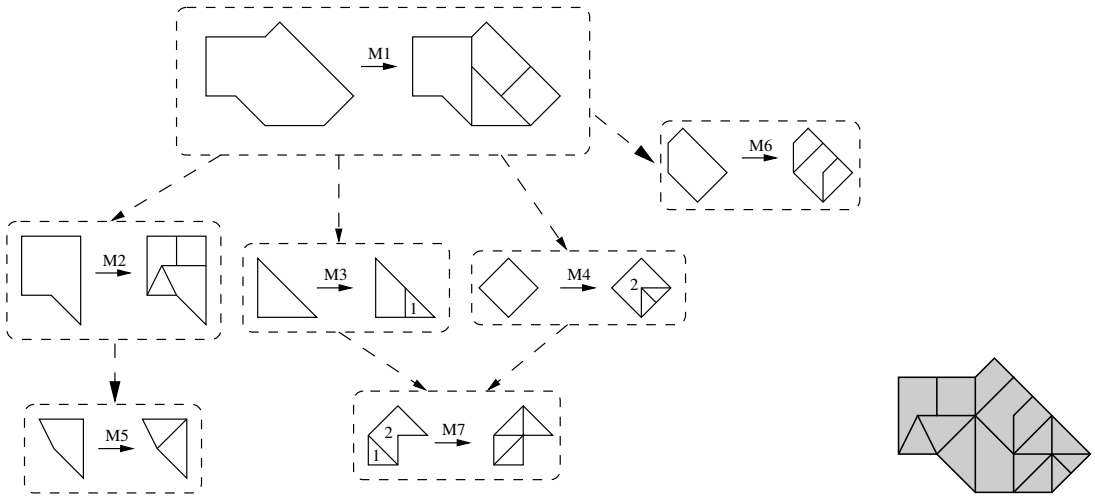


Figure 2.10 – *a. Exemple de graphe de De Floriani. - b. Objet obtenu après application de toutes les modifications du graphe (symbolisées par* $(M_i)_{i=1,...,7}$.

Dans la littérature, sont proposés deux autres modèles topologiques permettant de modéliser des objets complexes $3D$. Le premier est dédié à la modélisation de couches géologiques, le second plus général permet de modéliser des objets complexes quelconques, en utilisant différents mécanismes tels que la hiérarchie ou le clonage. Elles reposent toutes deux sur le modèle ordonné des *cartes généralisées* [Lie89].

Le premier modèle est issu du travail de Levy [Lev99] concernant la modélisation de couches géologiques. Levy ne représente de manière détaillée que les surfaces séparant des couches géologiques, les volumes des couches étant simplifiés. Il propose une hiérarchie de cartes généralisées à deux niveaux, nommée H-G-Carte. Le premier niveau représente les couches géologiques de manière simplifiée à l'aide d'une 3-G-carte. Chaque couche est définie par un polyèdre représentant son volume avec très peu de détail, et les couches sont liées entre elles par leurs relations d'adjacence. Dans le second niveau de la hiérarchie, sont représentés les détails des frontières, appelées *treillages*, à l'aide de 2-G-cartes. Le maillage séparant deux couches est lié aux deux faces simplifiées correspondantes dans le niveau supérieur. La figure 2.11 montre à gauche un exemple de couches géologiques et à droite une H-G-Carte : seulement 16 brins sont utilisés pour représenter la séparation entre deux couches au niveau supérieur de la hiérarchie (soit 8 brins pour chaque face) et le détail du treillage est décrit une seule fois dans le niveau inférieur (un pointeur met en correspondance les brins du niveau supérieur avec ceux du niveau inférieur). La hiérarchie permet d'avoir un maillage très précis au niveau des treillages sans pour autant les stocker deux fois en mémoire, une fois pour chaque face des deux volumes. Ce gain de mémoire est essentiel pour la modélisation de terrains géologiques complexes. Cette technique ne peut pas être directement adaptée aux complexes architecturaux. Nous souhaitons un plus grand nombre de niveaux dans la hiérarchie et une notion d'inclusion est nécessaire dans les bâtiments pour

indiquer quels meubles sont à l'intérieur de quelles pièces. De plus, ce modèle ne permet de représenter que des frontières séparant des volumes. Nous retenons néanmoins qu'une hiérarchie permet de diminuer l'espace mémoire nécessaire à une représentation et que ces opérations de construction et d'édition sont simplifiées par la hiérarchie, car elles se font de manière locale sur un petit nombre de faces. Nous avons conservé pour les bâtiments le lien explicite entre deux niveaux portant sur les éléments de base des modèles.

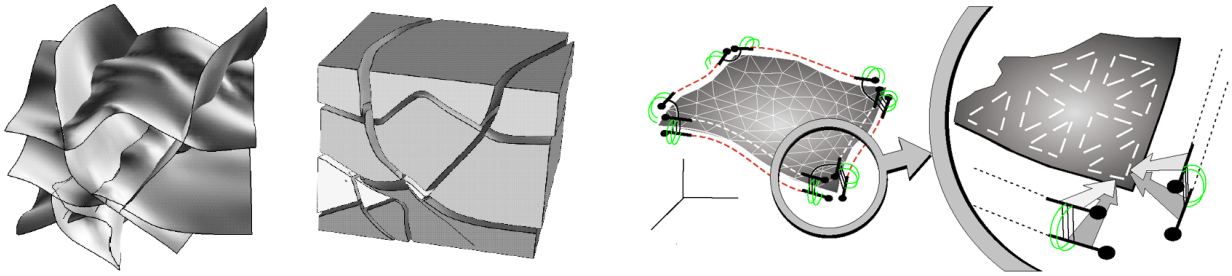


Figure 2.11 – *À gauche les limites entre les différentes couches géologiques à représenter, au centre les volumes de ces couches, à droite un schéma illustrant le principe des H-G-Cartes (images issues de la thèse de Levy).*

La seconde structure est celle des H-objets, proposée par Guilbert [Gui00]. Elle utilise un graphe orienté de cartes généralisées. Chaque noeud est une G-carte et les arcs expriment le *remplacement* d'un motif par un autre : son détail. Aux arcs sont associées les transformations (translations et rotations) nécessaires pour positionner un détail à la place de son parent. Puisqu'il s'agit d'un graphe, le modèle autorise plusieurs motifs pour un même détail. Ceci permet de *cloner* les objets. La figure 2.12 montre un exemple de H-Objet : plusieurs parties d'un clocher simplifié sont associées avec le même détail pour représenter un clocher plus complexe. En revanche, ce modèle n'autorise pas le fait qu'un motif soit associé à plusieurs détails différents. À l'aide de cette structure, Guilbert a construit manuellement plusieurs objets complexes : un arbre, un oeil humain, etc. Un exemple de bâtiment est aussi proposé dans sa thèse : l'église d'Orcival (figure 2.12 à droite). L'utilisation du clonage d'objets permet une économie mémoire non négligeable. Il s'agit néanmoins d'un modèle dans lequel il manque certaines informations topologiques puisque, si deux motifs sont liés, rien n'indique comment les détails le sont.

2.2.4 Discussion

Les structures proposées par De Floriani sont dédiées aux opérations de modification de maillage entre deux niveaux d'un LOD et doivent être évaluées pour calculer l'objet résultant. Ces modèles théoriques sont difficilement adaptables à la modélisation d'environnements architecturaux et à la problématique liée à la visualisation. Certaines opérations de modification (insertion de faces à un niveau donné, suppression d'une face, opérations booléennes) sont difficiles à implanter sur ce type de structures (par exemple, les opérations n'ont été définies qu'en $2D$ pour les maillages en multirésolution de De Floriani). De plus, l'évaluation d'une structure

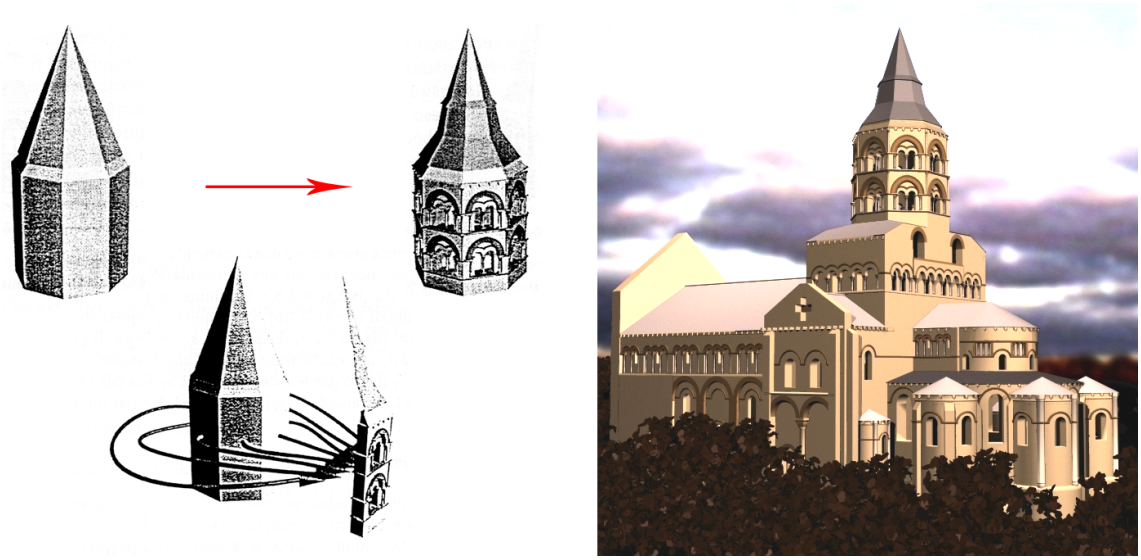


Figure 2.12 – *À gauche en bas, un exemple de H-Objet représentant un clocher à deux niveaux de détails, tous deux visibles au dessus. À droite, l'église d'Orcival modélisée par Guilbert (images issues de la thèse de Guilbert).*

est trop coûteuse pour permettre une visualisation efficace. Nous cherchons une représentation exploitable sans évaluation géométrique, et seul un modèle exhaustif le permet. L'exemple de la figure 2.10 peut être représenté par une hiérarchie explicite comme sur la figure 2.13. Le cas particulier de la double décomposition visible dans le graphe de De Floriani (transformations numérotées $M3$, $M4$ et $M7$ sur la figure 2.10) ne peut pas être décrit à l'aide un arbre, nous devons donc adjoindre deux partitions au niveau de ce détail : regroupements des six triangles en deux polygones ou en quatre triangles (entourés en pointillé sur la figure). Pour avoir la même puissance d'expression que les graphes de De Floriani, notre modèle doit permettre de représenter une hiérarchie et plusieurs partitions d'une même partie de l'objet.

Les modèles de Levy et de Guilbert sont basés sur une hiérarchie de structures topologiques ne nécessitant pas (Levy) ou peu (Guilbert) d'évaluations pour obtenir une description explicite de la géométrie d'un objet. Néanmoins, le modèle de Levy est dédié à la description des contours de couches géologiques en utilisant une hiérarchie à deux niveaux. Nous généralisons dans la suite cette idée pour modéliser d'autres types d'objets tels que les grands bâtiments. Le modèle de Guilbert, plus générique, repose sur des remplacements de motifs de n'importe quelle dimension. En particulier, il n'interdit pas de remplacer un objet de dimension 2 par un objet de dimension 3. Dans le cadre d'un modeleur de bâtiments, nous préférons assurer une cohérence topologique et géométrique entre deux niveaux de détail. D'une part, nous souhaitons que le détail d'une face soit contenu dans celle-ci, et donc qu'il soit une autre face ou un ensemble de faces coplanaires (maillage). D'autre part, un volume à un niveau de détail donné doit contenir l'ensemble de ses détails dans les niveaux inférieurs. Ceci doit permettre de faciliter les algorithmes de visualisation

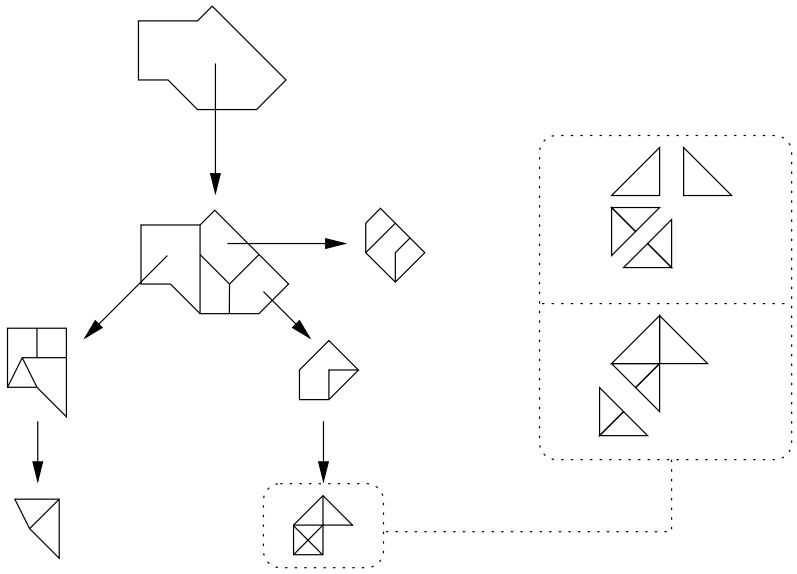


Figure 2.13 – *Exemple d'arbre utilisant une partition à un de ses niveaux pour exprimer les deux décompositions possibles d'un même polygone : en deux polygones (faces* 1 *et* 2 *de la figure précédente) ou en quatre triangles.*

à l'aide de règles simples : si une face/un volume est vue par l'observateur ou atteinte par un rayon, alors sa description plus détaillée le sera aussi.

Au vu de l'ensemble des données pouvant être représentées par des structures topologiques, nous avons choisi d'utiliser l'une d'elles comme base de notre structure de données. Pour des raisons que nous expliquons dans le chapitre suivant, nous avons décidé d'utiliser les cartes généralisées. Afin de pouvoir décrire des modèles architecturaux complexes, nous retenons de l'ensemble de ces modèles les principes de hiérarchisation, de clonage et de partitionnement des données. La hiérarchie permet de travailler localement sur différentes parties d'une scène complexe et de la structurer en fonction de l'importance des données modélisées à chaque niveau. Le fait d'utiliser des clones d'objets permet de limiter le nombre d'informations redondantes à l'intérieur du bâtiment : si une même chaise existe dans plusieurs pièces, une seule description existe en mémoire et les rotations et translations permettent de la placer en différents endroits. Enfin, les partitions permettent de compléter les informations de la hiérarchie, puisque comme nous l'avons vu précédemment la hiérarchie seule ne permet pas de représenter deux décompositions différentes d'un même détail.

Actuellement, les algorithmes de visualisation ne prennent pas en compte les structures topologiques ou géométriques précédemment citées. Les stratégies de construction de l'objet lors de la phase de modélisation sont ignorées. En effet, les structures de données issues de la modélisation sont souvent trop lentes pour permettre des algorithmes rapides de visualisation et de simulation d'éclairage. Elles sont optimisées pour les opérations de modifications et d'assemblage ; or les objets d'une scène en simulation d'éclairage sont en général géométriquement inchangés : l'observateur bouge, les objets se déplacent quelquefois dans l'espace, mais leur description

géométrique ne change pas. La plupart des techniques utilisent des listes de faces en données d'entrée de leurs algorithmes, puis calculent des informations géométriques et/ou topologiques pour structurer la scène. Le modèle que nous souhaitons construire doit donc aussi répondre à cet impératif de rapidité d'accès aux données et aussi à celui d'une occupation mémoire raisonnable afin de pouvoir y adjoindre les données nécessaires aux calculs de visualisation et de simulation d'éclairage.

CHAPITRE 3

NOTRE MODÈLE HIÉRARCHIQUE

Nous avons présenté dans le chapitre précédent les structures accélératrices communément utilisées en visualisation et les modèles topologiques plus complets exploités en modélisation. Ceci nous a permis de mettre en évidence un ensemble d'informations exploitées par les algorithmes des deux domaines, ainsi que les points forts de chaque structure. Rappelons que les informations très complètes issues de la phase de modélisation sont fréquemment perdues pour la phase de visualisation, alors qu'elles pourraient être utiles. Par exemple, de nombreux travaux [AB90, TS91, MBMD98] ont pour objectif de reconstruire à partir de la liste de faces décrivant un objet, des informations plus élaborées, utiles pour le rendu et disponibles en général lors de la phase de modélisation. C'est pourquoi nous proposons dans cette partie un modèle hiérarchique à base topologique permettant la modélisation de complexes architecturaux. Ce modèle a été conçu aussi en vue de son exploitation en visualisation et simulation d'éclairage.

Nous avons mis en évidence différentes notions qui nous semblent essentielles pour la modélisation des complexes architecturaux, ainsi que pour leur visualisation. Les informations topologiques sont indispensables pour faciliter la phase de modélisation et permettre certaines optimisations des algorithmes de visualisation. En effet, nous avons besoin de connaître un grand nombre d'informations concernant la scène telles que : quelle pièce est reliée à telle autre par une ouverture, quelle ouverture donne sur l'extérieur, à quelle pièce appartient telle face, etc. Ces informations utiles pour compléter les stratégies de visualisation sont essentiellement de nature topologique (relation d'incidence et d'adjacence entre cellules : sommets, arêtes, faces et volumes). La figure 3.1 illustre la subdivision représentant un bâtiment et montre premièrement l'intérêt d'une représentation volumique (par exemple, un mur a une épaisseur), et deuxièmement le besoin d'associer une certaine sémantique aux volumes de la subdivision (cette décomposition s'inspire des problèmes soulevés par les travaux en calcul de radiosité). Un deuxième point important concerne la structure du bâtiment (figure 1.4). Afin de représenter l'ensemble de ses agencements, il est nécessaire de représenter plusieurs partitions du bâtiment, de décrire des relations hiérarchiques ou d'inclusion entre différentes parties de la scène, etc. Les **partitions multiples** permettent par exemple de structurer le bâtiment en ailes ou en étages, de regrouper les sols et les murs, etc. L'utilisation d'une **hiérarchie** permet par exemple de représenter différents niveaux de précision et d'appliquer des traitements locaux plus rapides que des opérations influant sur la totalité du bâtiment. Il est ainsi possible de travailler sur des petites parties de la

scène, et de propager si nécessaire les opérations aux différents niveaux. Des informations sémantiques sont également utilisées pour différencier les volumes des murs de ceux des pièces par exemple. Des informations photométriques sont nécessaires pour la visualisation et la simulation d'éclairage. De plus, un maximum de données doit être explicité au sein du modèle pour éviter des évaluations coûteuses en temps de calcul. Enfin, une gestion minutieuse des informations devant résider en mémoire vive doit être réalisée pour conserver cette rapidité d'accès, et n'avoir en mémoire que l'ensemble des informations nécessaires aux calculs de modélisation ou de visualisation en cours.

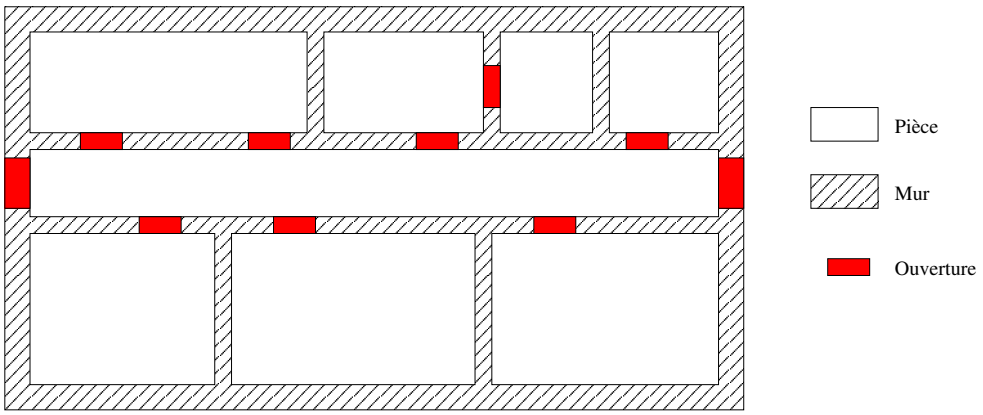


Figure 3.1 – *Le bâtiment est composé de différents volumes : les murs, les pièces et les ouvertures.*

Nous avons choisi d'enrichir le modèle des cartes généralisées. Nous définissons à partir de cette structure topologique deux mécanismes permettant de faciliter la modélisation et la visualisation : les *partitions multiples* et les *niveaux de détail*. Le premier permet à l'utilisateur de créer des groupes d'objets à l'intérieur des bâtiments, par exemple pour le structurer en ailes, en étages, ou pour regrouper les pièces suivant leur utilité (bureaux, salles de cours, etc). Le second s'approche des structures de niveaux de détails (LOD) souvent employés en visualisation pour réduire le nombre de primitives géométriques à traiter en fonction du point de vue. Ce mécanisme est également utile pour faciliter la construction d'un environnement complexe, comme nous le verrons dans le chapitre 4.

3.1 Les cartes généralisées

Les cartes généralisées [Lie89] sont définies mathématiquement comme des algèbres, à l'instar des algèbres d'arêtes [GS85] et des structures cell-tuples [Bri89]. Elles permettent de représenter la topologie de quasi-variétés subdivisées orientables ou non, avec ou sans bord [Lie94]. De manière informelle, les quasi-variétés constituent une classe d'objets intermédiaire entre les pseudo-variétés et les variétés, bien connues en topologie algébrique [Ago76].

De manière intuitive, une quasi-variété de dimension n peut être construite par assemblage de cellules de dimension n le long de cellules de dimension $n-1$, de façon à ce qu'une $(n-1)$-cellule

soit incidente à au plus deux n-cellules. En particulier, les cartes généralisées de dimension 3 permettent de représenter des subdivisions de $\mathbb{R}^3$, composées de volumes. Cette notion de volume est nécessaire ici pour décrire la structure du bâtiment : murs, pièces, etc.

3.1.1 Principe et définitions

Une carte généralisée est définie par un type unique d'éléments de base, appelés *brins*, sur lesquels sont définies des applications qui satisfont plusieurs propriétés énoncées dans la suite. Ceci permet de contrôler la cohérence topologique des objets modélisés. De nombreuses opérations ont été définies pour manipuler les cartes généralisées :

- les opérations de construction élémentaires ou élaborées, inspirées souvent d'opérations classiques et les généralisant parfois : identification (ou couture), suppression et contraction de cellules, triangulation, extrusion, produit cartésien, opérations booléennes, etc. ([Ber92, Elt94, Ska01, LMA+01, DL03]).
- les opérations de calcul de propriétés topologiques : calcul de bord, de l'orientabilité, du genre pour les 2-G-cartes, etc.

Du fait de la simplicité (un seul type d'éléments de base) et de l'homogénéité (en toute dimension) de leur définition, les G-cartes et les opérations de manipulation peuvent être implantées très simplement. Les définitions suivantes sont extraites de [Lie89].

Définition 1 *Soit n $\geq$ 0. Une carte généralisée de dimension n (ou n-G-Carte) est un $(n+2)$-uplet G=(B,α_0,α_1,...,α_n) où :*

- *B est un ensemble fini de brins,*
- *α_0, α_1, ..., α_n sont des applications telles que :*
  - *$\forall\ 0 \leq i \leq n$, α_i est une involution*[1],
  - *pour $\forall\ 0 \leq i < i+2 \leq j \leq n$, $\alpha_i \circ \alpha_j$ est une involution*[2].

La figure 3.2 montre une 3-G-carte composée de deux volumes (un cube et une pyramide) reliés par une face. Les cellules d'une carte généralisée sont définies implicitement comme des *orbites* [3] relativement à un sous-ensemble d'involutions.

Définition 2 *Soit $G = (B, \alpha_0, \alpha_1, \ldots, \alpha_n)$ une n-G-carte, et un brin $b \in B$. La i-cellule incidente au brin b est définie comme l'orbite $< \alpha_0, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_n > (b)$, notée $<>_{N\setminus\{i\}}(b)$ avec $N = [0, n]$.*

Pour i et b donné, la i-cellule incidente à b est donc formée par l'ensemble des brins pouvant être atteints à partir de b pour une composition quelconque d'involutions, α_i exceptée. La figure

[1] *Une bijection f est une involution si et seulement si $f \circ f$ est la fonction identité, i.e. $f = f^{-1}$.*

[2] *Dans la suite, nous utilisons pour des raisons de lisibilité des définitions une autre notation pour les applications : $b\alpha_i$ pour $\alpha_i(b)$ et $\alpha_j\alpha_i$ pour $\alpha_i \circ \alpha_j$.*

[3] *Soit $\Phi = \{\phi_i, 0 \leq i \leq n\}$ un ensemble de permutations définies sur un ensemble fini E, et soit $e \in E$. Notons $< \Phi >$ le groupe de permutations engendré par Φ. L'orbite de e relativement à Φ, notée $< \Phi > (e)$, est l'ensemble $\{e' = \phi(e), \phi \in < \Phi >\}$. De manière informelle, cette orbite est l'ensemble de tous les éléments de E pouvant être atteints à partir de e par une composition quelconque des éléments de Φ.*

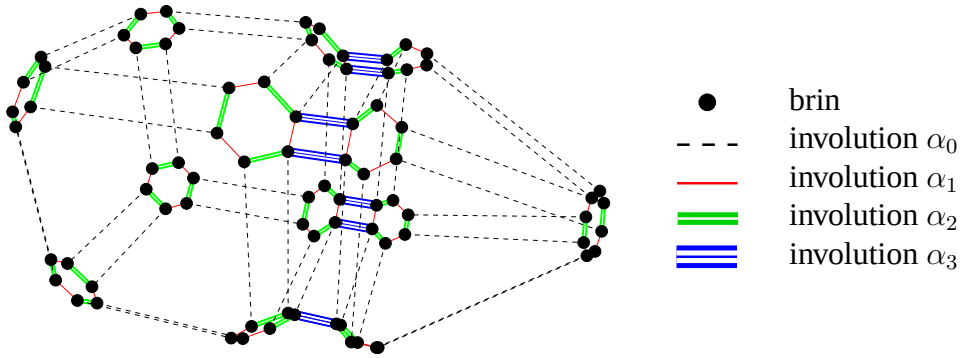


Figure 3.2 – *Représentation éclatée d'une 3-G-carte : nous obtenons un graphe où les nœuds représentent les brins et les arêtes les liaisons α.*

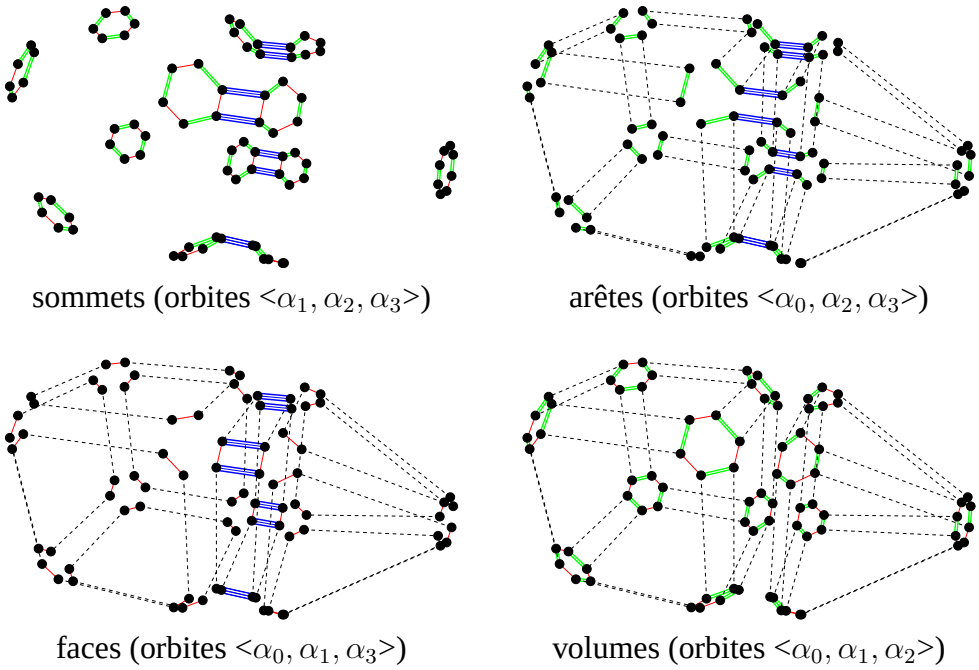


Figure 3.3 – *Mise en évidence des différentes cellules d'une 3-G-carte.*

3.3 montre les différentes i-cellules de la 3-G-carte de la figure 3.2. De la même manière, la composante connexe incidente à un brin b est définie comme l'orbite $<>_N\ (b) = <\alpha_0, \ldots, \alpha_n > (b)$, c'est-à-dire l'ensemble des brins pouvant être atteints à partir de b par une composition quelconque des involutions $\alpha_0, \ldots, \alpha_n$.

Dans la suite, nous utilisons différentes représentations graphiques pour un brin (figure 3.4.a), en fonction de "l'éclatement" avec lequel est dessinée la carte généralisée, et de la cellule portée par le brin que nous souhaitons mettre en évidence (le sommet, l'arête ou/et la face). Il ne s'agit que de conventions d'affichage permettant de mettre en évidence certaines informations topo-

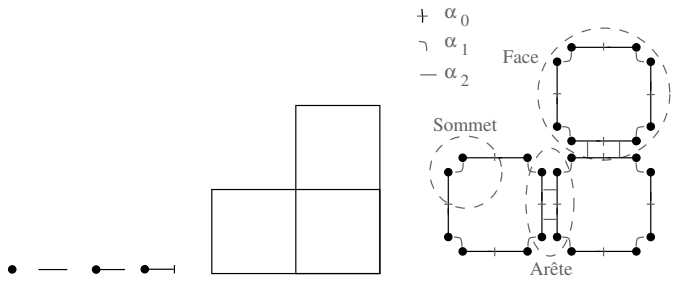


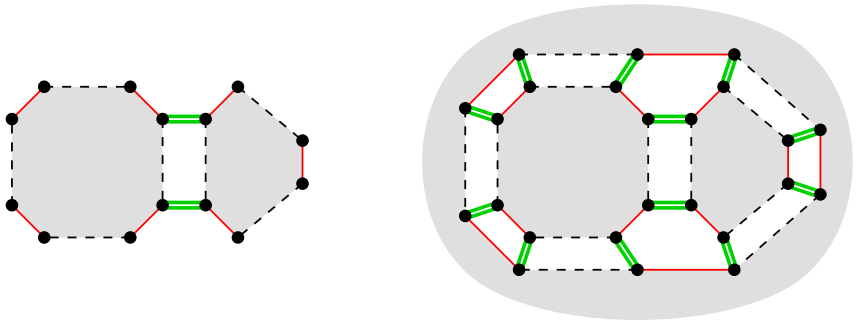
Figure 3.4 – *a. Différentes représentations d'un brin. - b.* 2*-G-carte affichée en fil de fer (affichage non éclaté). - c. La même* 2*-G-carte, en affichage éclaté, avec ses trois types de cellules mises en évidence : une face (orbite* $< \alpha_0, \alpha_1 >$*), une arête (orbite* $< \alpha_0, \alpha_2 >$*) et un sommet (orbite* $< \alpha_1, \alpha_2 >$*).*

logiques. Cet affichage est paramétré par un pourcentage d'éclatement. Si la carte généralisée est dessinée avec un taux d'éclatement nul, les involutions α ne sont pas mises en évidence. La figure obtenue ressemble à un affichage fil de fer, tous les brins d'une arête sont confondus en un segment (figure 3.4.b). Si l'affichage est semi-éclaté (éclatement entre 0 et 1), le brin apparaît alors comme une portion de segment reliée à un autre brin par l'involution α_0. Enfin, si l'éclatement est à son maximum, le brin est symbolisé par un sommet comme précédemment pour les figures 3.2. Le choix d'affichage que nous prenons par la suite dépend des informations que nous souhaitons mettre en valeur. Ce type d'affichage est implanté dans le modeleur que nous avons conçu, et s'inspire des travaux de Bertrand [Ber92].

Le brin est une notion abstraite pouvant s'expliquer intuitivement : dans une 3-G-carte, un brin est l'un des sommets d'une arête appartenant à une face appartenant elle-même à un volume (ceci correspond à la notion de cell-tuples proposée par Brisson [Bri89]). Un brin identifie donc l'ensemble des cellules qui lui sont incidentes. Cette définition d'une subdivision à partir d'un type unique d'éléments est l'un des principaux atouts des cartes généralisées. Ceci permet de définir des opérations élémentaires travaillant localement sur un brin, bases d'opérations topologiques plus complexes.

3.1.2 Opérations de construction

Une n-G-carte peut être construite à l'aide de deux opérations de base. La première d'entre elles est l'ajout de brins à l'ensemble B. Ces brins sont alors définis comme invariants pour toutes les involutions : au départ, un nouveau brin n'est jamais lié à un autre. La deuxième opération consiste à assembler des cellules : la *couture* (ou *identification*). Par exemple, dans une 3-G-carte, la couture de deux volumes (orbite $<\alpha_0, \alpha_1, \alpha_2>$) par α_3 permet d'assembler deux volumes en identifiant deux de leurs faces. Pour ce faire, les deux faces (orbite $<\alpha_0, \alpha_1>$) doivent avoir la même structure, i.e. plus formellement être isomorphes (par exemple, le cube et la pyramide de la figure 3.2 ne peuvent être assemblés que le long de faces carrées). Plus généralement, la couture de deux cellules de dimension i (orbites $<>_{N\setminus\{i\}}$) par α_i se définit ainsi :



a b

Figure 3.5 – *a. Une 2-G-carte ayant un bord, elle ne subdivise qu'une partie de l'espace. - b. Une 2-G-carte sans bord subdivisant l'espace* $\mathbb{R}^2$.

Définition 3 *Soit* $G = (B, \alpha_0, \alpha_1, \ldots, \alpha_n)$ *une* n*-G-carte, et* $b, b' \in B$*, deux brins tels que* b *et* b' *sont invariants par* α_i*. La couture par* α_i *de* b *et* b' *peut être réalisée si et seulement si* $<>_{N\setminus\{i\}}(b)$ *est isomorphe à* $<>_{N\setminus\{i\}}(b')$*. Soit* ϕ *cet isomorphisme. Le résultat de cette opération est une* n*-G-carte* $G' = (B, \alpha_0, \ldots, \alpha_{i-1}, \alpha'_i, \alpha_{i+1}, \ldots, \alpha_n)$ *telle que :*

- $\forall\, b'' \in <>_{N\setminus\{i\}}(b)$ *et* $b''' \in <>_{N\setminus\{i\}}(b')$, $b''\alpha'_i = b''\phi$ *et* $b'''\alpha'_i = b'''\phi^{-1}$,
- $\forall\, b'' \notin <>_{N\setminus\{i\}}(b) \cup <>_{N\setminus\{i\}}(b')$, $b''\alpha'_i = b''\alpha_i$.

Une autre particularité des G-cartes est de pouvoir représenter des subdivisions avec ou sans bord (voir figure 3.5).

Définition 4 *Soit* $G = (B, \alpha_0, \alpha_1, \ldots, \alpha_n)$ *une* n*-G-carte.* G *est sans bord si et seulement si* $\forall\, i \in [0, n]$ *et* $\forall\, b \in B, b\alpha_i \neq b$.

Par la suite, comme nous voulons modéliser des subdivisions de l'espace $\mathbb{R}^3$, nous utilisons des 3-G-cartes sans bord.

Les cartes généralisées ne permettent de représenter que la topologie d'un objet, et non sa forme géométrique. Sans information supplémentaire, une G-carte ne peut pas être affichée, car ses cellules ne sont pas positionnées dans l'espace. Pour décrire la forme de l'objet, des informations sont associées aux cellules de la G-carte[4]. Ces informations portent le nom de *plongements* ou d'*attributs*. Elles peuvent être de différentes natures : coordonnées spatiales, couleurs, vecteurs, etc. Par exemple, pour définir la forme d'une 3-G-carte, nous associons à chacun des sommets (orbites $<\alpha_1, \alpha_2, \alpha_3>$) des coordonnées cartésiennes. Les objets manipulés ici sont polyédriques, les arêtes sont donc des segments reliant les sommets et il n'est pas nécessaire de décrire explicitement la géométrie des 1-cellules. Cependant, il serait possible d'associer (en plus des coordonnées dans les sommets) des courbes aux arêtes (orbites $<\alpha_0, \alpha_2, \alpha_3>$) et des surfaces quelconques aux faces (orbites $<\alpha_0, \alpha_1, \alpha_3>$).

[4] *En pratique, comme toute orbite est identifiée par la donnée d'un brin et de ses involutions, ces informations sont en général associées aux brins. Pour éviter les redondances, les informations peuvent n'être associées qu'à un unique brin de l'orbite.*

3.2 Étiquetage de cartes généralisées

Nous souhaitons pouvoir structurer une carte généralisée pour décrire plusieurs partitions d'un même objet, par exemple pour grouper dans un bâtiment les pièces, les murs et les ouvertures en ailes ou en étages. Nous avons donc besoin de définir des mécanismes permettant de grouper des cellules et de caractériser ces groupes, en leur associant des informations communes (sémantique du groupe, informations photométriques ou géométriques, etc.). De la même manière, nous voulons structurer le bâtiment, en le représentant par une hiérarchie de cartes généralisées, afin de permettre un travail local et un chargement partiel de certaines parties du bâtiment (par exemple pour ne pas avoir tous les meubles en mémoire vive, pour créer des portes et des fenêtres sans être obligés de charger tous les étages en mémoire vive, etc.). Cette hiérarchie de cartes généralisées peut être définie en utilisant la notion de groupes d'objets. Par exemple, une pièce est un groupe constitué des murs de la pièce, de ses ouvertures et de ses meubles ; un étage est un groupe de pièces (donc un groupe de groupes) et le bâtiment est un groupe d'étages.

À l'aide de la notion de groupe, nous pouvons donc définir conjointement la multipartition et la hiérarchie. Dans un premier temps, nous avons défini les groupes de manière abstraite par un étiquetage, c'est-à-dire en associant à chaque cellule (et donc à chaque brin) un ou plusieurs numéros de groupes. Nous détaillons dans cette section les définitions de la hiérarchie et des partitions multiples à l'aide de fonctions de numérotation particulières.

Notons que ce modèle n'a pas été mis en œuvre de cette manière. Les définitions suivantes nous servent néanmoins de base pour la conception d'optimisations et pour assurer la cohérence du modèle final présenté dans la section 3.3. Même s'il représente une solution théorique pour la définition de notre structure, il ne répond pas à certains impératifs fixés, en particulier liés à l'encombrement mémoire.

3.2.1 Étiquetage pour la multipartition

L'un de nos objectifs pour la modélisation est de regrouper des objets de même sémantique (nom, utilité, matériau, etc.). Pour cela, étiqueter les cellules consiste à leur affecter l'identifiant du groupe auquel elles appartiennent. Pour grouper des cellules de dimension n, (des étages ou des pièces en dimension 3 par exemple), nous définissons une fonction de partition ϕ_n associant à chaque cellule son identifiant. De la même manière, en généralisant le concept de groupe aux cellules de dimensions inférieures (grouper les faces d'un maillage, les arêtes alignées par exemple), nous pouvons définir une fonction de numérotation ϕ_i pour les i-cellules. Dans le cas des cartes généralisées, cette fonction ϕ_i peut être définie sur les brins, en respectant les contraintes de cohérence précisant que deux brins d'une même i-cellule ont même étiquetage.

Définition 5 : étiquetage de i-cellules
Soient $G = (B, \alpha_0, ..., \alpha_n)$ une carte généralisée de dimension n et $\phi_i : B \longrightarrow \mathbb{N}$, une fonction de numérotation des brins. ϕ_i est une fonction de partition de G en groupes de i-cellules si et seulement si $\forall\, b \in B, \forall b' \in <>_{N\setminus\{i\}}(b),\ \phi_i(b) = \phi_i(b')$.

La figure 3.6 montre un exemple d'étiquetage de 2-cellules (faces) pour une 2-G-Carte. Cette figure représente la même partition de deux manières différentes. À gauche, le numéro de chaque

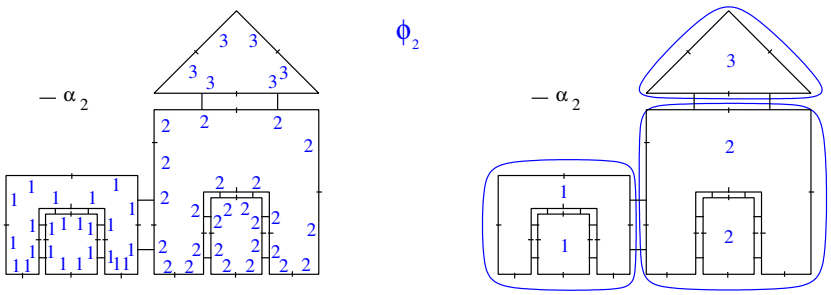


Figure 3.6 – *Exemple de partition d'une carte généralisée à l'aide d'un étiquetage des 2-cellules par la fonction ϕ_2. Nous représentons ici le dessin très simple d'une maison subdivisée en trois groupes principaux : l'habitat, le toit et le garage. À gauche, chaque brin est numéroté par le numéro de sa partition. À droite, seules les cellules sont numérotées pour une meilleure compréhension du schéma.*

brin est affiché à son côté. Cette notation étant peu lisible, dans la suite, les étiquettes seront associées aux cellules concernées comme à droite de la figure.

Plusieurs fonctions de numérotation peuvent être définies pour une même carte généralisée et pour des valeurs quelconques de i (i $\in [0, n]$). Ceci permet donc de décrire plusieurs partitions sur une même carte généralisée, et ainsi de définir la notion de multipartition (voir figure 1.4.a). La définition suivante ne spécifie aucune contrainte supplémentaire, les différentes partitions de l'objet étant *a priori* indépendantes les unes des autres.

Définition 6 : étiquetage d'une multipartition
Soient $G = (B, \alpha_0, ..., \alpha_n)$, une carte généralisée de dimension n, et P fonctions de partition $(\phi^p_{i_p})_{p\in\{1..P\}, i_p\in[0,n]}$ de G. L'ensemble de ces fonctions de partitions $\Phi = \{\phi^p_{i_p}\}_{p\in\{1..P\}}$ forme une multipartition de G.

3.2.2 Étiquetage pour la hiérarchie

Comme nous l'avons dit précédemment, nous souhaitons aussi structurer un bâtiment de manière hiérarchique. La racine de la hiérarchie correspond au niveau le plus simple, les niveaux suivants détaillent les niveaux précédents (par convention, nous disons qu'un niveau est supérieur à un autre s'il est plus proche de la racine, i.e. s'il est moins détaillé).

Cette notion de hiérarchie ne correspond pas exactement à la notion de niveau de détail (LOD) présenté en section 2.1.1. En effet, nous souhaitons imposer des contraintes topologiques et géométriques entre niveaux consécutifs de la hiérarchie. Les contraintes sont nécessaires pour l'optimisation de certains algorithmes : par exemple, si une partie d'objet décrite à un niveau est détaillée dans un niveau inférieur, le détail est inclus géométriquement dans la partie d'objet. Par exemple, pour un algorithme de tracé de rayons, ceci permet de garantir que si un rayon n'a pas d'intersection avec une partie de l'objet, il n'en a pas non plus avec ses détails. Cette propriété permet donc d'optimiser les calculs d'intersection.

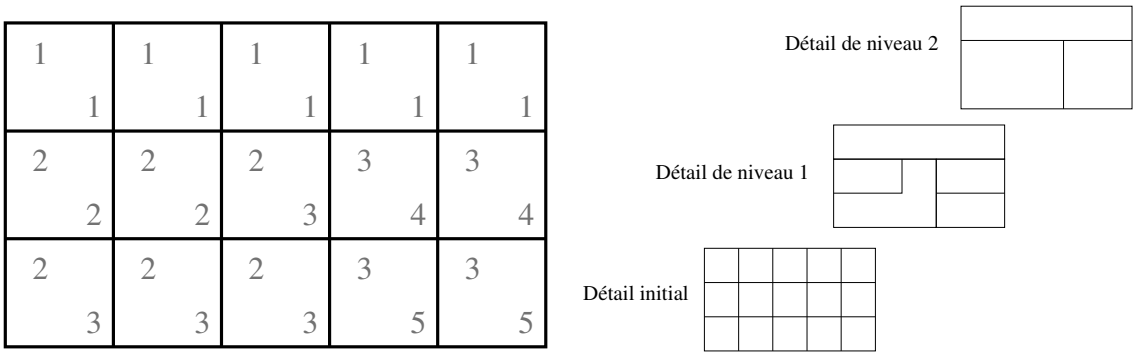


Figure 3.7 – *a. Une hiérarchie à 3 niveaux définie par 2 fonctions de partitions ϕ_2^1 et ϕ_2^2. L'étiquetage ϕ_2^1 (dans le coin inférieur droit) correspond à un niveau de détail intermédiaire entre le détail maximal et le niveau de détail le moins précis représenté par ϕ_2^2 (affiché dans le coin supérieur gauche des 2-cellules). - b. Simplifications correspondant à ces étiquetages.*

Le mécanisme d'étiquetage permet de représenter une hiérarchie de de type, à condition de définir des contraintes de cohérence sur les étiquettes. Supposons par exemple que nous manipulons une G-carte représentant un objet. Le niveau supérieur de la hiérarchie est décrit par cette même G-carte dans laquelle certaines n-cellules ont été regroupées, en utilisant le mécanisme des partitions décrit dans la section précédente. Nous pouvons continuer à structurer cette G-carte en regroupant au niveau encore supérieur des groupes du second niveau (voir figure 3.7). Chaque groupe d'un niveau donné d est donc un regroupement de groupes du niveau inférieur $d-1$ (le principe est ici très proche de celui des pyramides d'images, où une région d'un niveau donné est un ensemble de régions du niveau directement inférieur).

Pour les cartes généralisées, ce principe est traduit en terme d'étiquetage de brins (en supposant pour simplifier que seuls les cellules de dimension n sont groupées) de la manière suivante :

- chaque niveau k de la hiérarchie est défini par une fonction de partition ϕ_n^k de la G-carte en groupes de n-cellules ;
- si deux brins font partie du même groupe au niveau k, ils font partie du même groupe pour chacun des niveaux supérieurs à k.

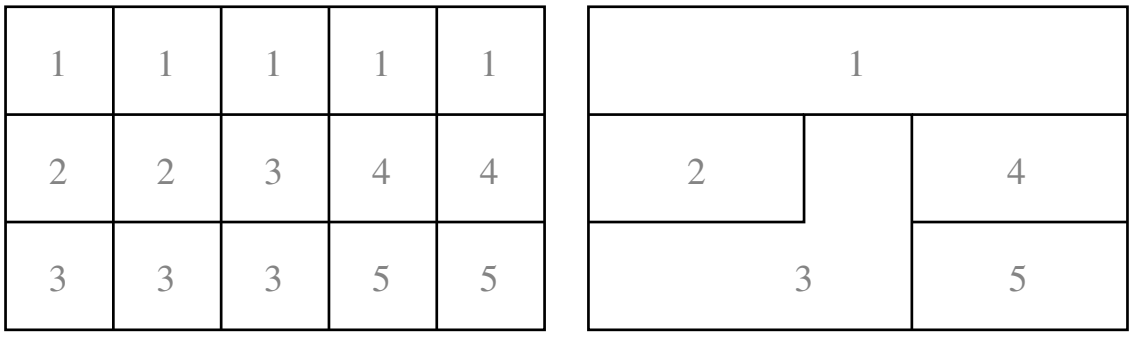
Plus formellement, nous obtenons la définition suivante :

Définition 7 *Soit $G = (B, \alpha_0, ..., \alpha_n)$, une n-G-carte. Une hiérarchie sur G est définie par une famille de D fonctions de partitions $(\phi_n^d)_{d\in[1,D]}$ vérifiant,*

- $\forall d > 1,\ \phi_n^d \neq \phi_n^{d-1}$;
- $\forall d > 1, \forall b, b' \in B,\ \phi_n^{d-1}(b) = \phi_n^{d-1}(b') \Rightarrow \phi_n^d(b) = \phi_n^d(b')$.

Le couple (G, ϕ_n^{d-1}) est appelé détail du couple (G, ϕ_n^d).

Pour un niveau de détail d donné, une G-carte simplifiée peut être déduite de la G-carte initiale et de la fonction de partition ϕ_n^d, en supprimant les cellules de dimension $n-1$ incidentes à deux cellules de dimension n appartenant au même groupe (voir figure 3.8).



a. b.

Figure 3.8 – *a. Un objet dont les 2-cellules (faces) sont groupées par une fonction de partition ϕ_2. - b. Simplification déduite de cet étiquetage.*

L'opération de suppression de cellules de dimension quelconque a été récemment revisitée par Damiand *et al.* [DL03] (voir aussi [Elt94]). Ils proposent une définition très simple de la suppression de cellules de degré[5] 2. La définition suivante est illustrée par la figure 3.9.

Définition 8 *Soient $G = (B, \alpha_0, ..., \alpha_n)$, une n-G-carte. Soit $i \in [0, n-1]$. Soit b un brin de B tel que la i-cellule $C =<>_{N\setminus\{i\}} (b)$ vérifie[6] : $\forall b' \in C$, $b'\alpha_{i+1}\alpha_{i+2} = b'\alpha_{i+2}\alpha_{i+1}$.*
Notons $B^S = C\alpha_i \setminus C$, l'ensemble des brins voisins par α_i des brins de C et n'appartenant pas à C. La n-G-carte $G' = (B', \alpha'_0, ..., \alpha'_n)$ résultant de la suppression de C est définie par :

- $B' = B \setminus C$;
- $\forall j \in \{0, \ldots, n\} \setminus \{i\}$, $\alpha'_j = \alpha_j \mid_{B'}$;
- $\forall b' \in B' \setminus B^S, b'\alpha'_i = b'\alpha_i$;
- $\forall b' \in B^S, b'\alpha'_i = b'(\alpha_i\alpha_{i+1})^k\alpha_i$, *où k est le plus petit entier positif tel que* $b'(\alpha_i\alpha_{i+1})^k\alpha_i \in B^S$.

Cette définition est généralisée dans [DL03] en une opération permettant de supprimer simultanément des cellules de dimension quelconque, à condition que :

- les cellules à supprimer soient disjointes, i.e. deux cellules quelconques n'aient aucun brin en commun ;
- les cellules à supprimer soient toutes de degré 2.

En utilisant cette opération plusieurs fois, il est possible néanmoins de supprimer des cellules de degré supérieur à 2. Pour cela, une succession d'opérations de suppression est nécessaire pour supprimer des cellules incidentes. Par exemple, pour supprimer dans une 2-G-carte (figure 3.9.a), un sommet de degré 4 (sommet à l'intérieur du maillage de l'escalier), il faut tout d'abord

[5] *De manière schématique, un sommet de degré 2 est incident à deux arêtes, et plus généralement une i-cellule de degré 2 est incidente à deux $(i+1)$-cellules.*

[6] *Cette condition correspond schématiquement au fait que la cellule est de degré 2. Notons que si $i = n-1$, la condition est toujours vérifiée, car par définition des G-cartes, toute $(n-1)$-cellule est au plus de degré 2 et il est toujours possible de supprimer toute $(n-1)$-cellule.*

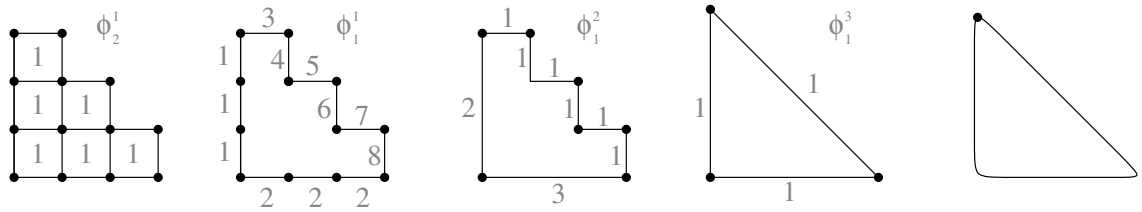


Figure 3.9 – *a. Détail d'un escalier (précision maximale) et fonction d'étiquetage ϕ_2^1 des faces définissant le niveau de détail suivant. - b. Simplification intermédiaire mettant en évidence l'étiquetage ϕ_1^1 des arêtes. - c. Niveau de détail déduit des étiquetages précédents (ϕ_2^1,ϕ_1^1), et nouvel étiquetage des arêtes pour la simplification suivante ϕ_1^2 - d. Détail le plus simple de l'escalier déduit de ϕ_1^2. - d. Détail de l'escalier déduit de ϕ_1^2. - e. Simplification déduite de ϕ_1^3.*

supprimer deux des arêtes qui lui sont incidentes pour le rendre de degré 2. En règle générale, la suppression d'un i-cellule de degré supérieur à 2 dans une n-G-carte implique la suppression de certaines $(n-1)$-cellules, $(n-2)$-cellules, ..., $(i+1)$-cellules incidentes pour que cette i-cellule devienne de degré 2.

En nous basant sur cette définition générale, nous pouvons étendre la notion de hiérarchie de G-cartes précisée par la définition 7, en permettant de grouper entre deux niveaux consécutifs des groupes satisfaisant aux conditions de la définition 8 (et en particulier des cellules de dimension quelconque satisfaisant à ces conditions, figure 3.9).

3.3 Modèle final : une hiérarchie de multipartitions

L'étiquetage est un mécanisme théorique simple permettant de définir un modèle intégrant à la fois les notions de multipartition et de hiérarchie. Les contraintes imposées par la gestion de la mémoire et les temps de calcul doivent être prises en compte, car la représentation d'un bâtiment meublé nécessite le stockage d'un nombre très important d'informations topologiques (brins, partitions et hiérarchie) et géométriques (plongements : sommets, BRDF, textures, etc). Un modèle topologique se fondant sur l'étiquetage exige une grande quantité de mémoire, puisque chaque brin doit être étiqueté par un entier pour chaque partition et chaque niveau de hiérarchie. De plus, l'étiquetage induit trop de redondances inutiles qui limitent son utilisation en pratique.

Pour garantir une occupation mémoire raisonnable et conserver l'efficacité des opérations de base, en particulier les parcours, nous proposons une représentation optimisée du modèle précédent. Nous proposons de séparer les représentations de la multipartition et de la hiérarchie, car comme nous le verrons par la suite, nous utilisons ces notions à des fins différentes dans le modeleur de bâtiments, et nous les manipulons par des ensembles d'opérations distinctes.

Les différents choix réalisés sont motivés à chaque étape par un souci d'efficacité en termes de visualisation. La structure finale n'est donc pas uniquement dédiée à la modélisation ; elle contient également des informations utiles à une accélération des calculs de visualisation.

Pour ces raisons, nous avons choisi d'optimiser le modèle précédent en utilisant différentes techniques :

- concernant la multipartition, la plupart des besoins de regroupement de cellules que nous avons relevés se rapportent à des ensembles connexes de cellules de dimension n. Nous avons donc choisi de représenter ces groupes par un marquage des liaisons α_n ; il reste possible de grouper des ensembles de cellules de dimension n non connexes, ou des ensembles de cellules de dimension inférieure, en utilisant les fonctions de partitions.
- concernant la hiérarchie, nous avons choisi de représenter, schématiquement parlant, la "différence" entre un niveau et le niveau précédent de la hiérarchie, afin de réduire la redondance des informations.

3.3.1 Les multipartitions à l'aide d'un marquage

L'étiquetage permet de tester très rapidement si deux cellules appartiennent au même groupe. Néanmoins, en pratique, étiqueter une carte généralisée est coûteux. En effet, il faut attribuer un numéro unique à chaque groupe d'une partition existante et l'affecter à chacun des brins. À chaque brin, sont attribués autant d'identifiants de groupe que de partitions, et chaque numéro est représenté par un entier. Pour diminuer le coût mémoire, nous proposons une optimisation qui ne permet de créer que des groupes de cellules de dimension n connexes, et qui autorise aussi des parcours plus rapides.

D'un point de vue technique, une G-carte peut être considérée comme un graphe dont les noeuds correspondent aux brins et dont les arêtes lient deux nœuds si les brins correspondants sont "liés par une involution α" (i.e. sont image l'un de l'autre par une involution α). Une composante connexe d'une G-carte est alors une composante connexe de ce graphe. Pour représenter des sous-graphes connexes, nous pouvons distinguer les arêtes des sous-graphes par une marque booléenne.

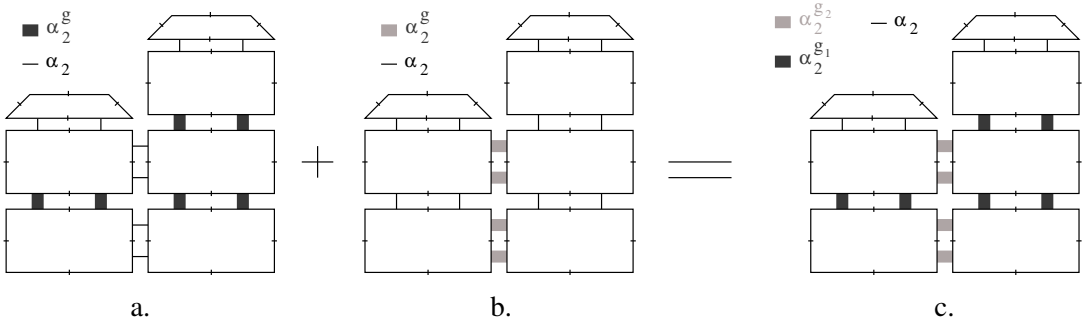


Figure 3.10 – *Liaisons groupantes : à gauche sont représentées deux partitions différentes d'un même bâtiment. - a. Une première décomposition en ailes - b. Une seconde en étages. - c. La G-Carte bi-partitionnée correspondante.*

Plus précisément, si nous voulons représenter des groupes constitués d'ensembles connexes de cellules de dimension n, nous devons exprimer si deux cellules font partie ou non du même

groupe (voir figure 3.10) à l'aide d'un marquage par des booléens associés aux liaisons α_n. Plus formellement, nous donnons la propriété suivante :

Définition 9 : les liaisons groupantes
Soit une carte généralisée $G = (B, \alpha_0, ..., \alpha_n)$ et une fonction de partition ϕ_n définie sur G. ϕ_n définit une partition de G en groupes connexes si et seulement si :
- $\forall b_1, b_2 \in B$, *tels que* $\phi_n(b_1) = \phi_n(b_2)$, *il existe* $i_1, ..., i_p \in [0, n]$ *tels que* $b_2 = b_1\alpha_{i_1} \dots \alpha_{i_p}$;
- *et* $\forall j, 1 \leq j \leq p, \phi_n(b_1\alpha_{i_1} \dots \alpha_{i_j}) = \phi_n(b_1)$.

Dans le cas où ϕ_n définit une partition de G en groupes connexes, l'involution groupante α_n^g est définie par restriction de α_n aux groupes définis par ϕ_n, c'est-à-dire $\forall b_1, b_2 \in B$ tels que $\alpha_n(b_1) = b_2$, nous posons :
- *si* $\phi_n(b_1) = \phi_n(b_2)$ *alors* $\alpha_n^g(b_1) = b_2$ *et* $\alpha_n^g(b_2) = b_1$;
- *sinon* $\alpha_n^g(b_1) = b_1$ *et* $\alpha_n^g(b_2) = b_2$.

En pratique, comme α_n^g est une restriction de α_n qui ne peut prendre que deux valeurs possibles pour deux brins images l'un de l'autre par α_n, α_n^g peut être représentée par l'association d'une marque booléenne à toute liaison α_n. Cette optimisation correspond néanmoins à une restriction, puisqu'elle ne permet de représenter que des groupes de cellules connexes. De plus, elle implique un parcours coûteux pour déterminer l'appartenance de deux brins à un même groupe.

Pour expliquer ce choix, il faut noter que des groupes de cellules non connexes n'ont que peu d'utilité pour notre domaine d'application. En effet, les groupes servent dans le modeleur, d'une part à maintenir une cohérence dans la hiérarchie (puisque nous avons vu que la hiérarchie correspond à une multipartition particulière) et d'autre part pour la visualisation, afin de regrouper des pièces presque toujours connexes. Néanmoins, pour conserver la possibilité de créer des groupes de cellules non connexes, des attributs de groupe peuvent être associés aux cellules. Ainsi en ajoutant un attribut identique à deux groupes non connexes, nous pouvons indiquer qu'ils forment un même groupe. Enfin, nous avons vu que plusieurs étiquetages peuvent coexister pour décrire plusieurs partitions. De la même manière, plusieurs marquages de liaisons permettent de définir une multipartition.

Définition 10 : les multipartitions
Soient une carte généralisée $G = (B, \alpha_0, ..., \alpha_n)$ et p fonctions ϕ_n^1, ..., ϕ_n^p de partition de G en groupes connexes. Pour chaque fonction ϕ_n^k, $1 \leq k \leq p$, l'involution $\alpha_n^{g_k}$ est définie par, $\forall b \in B$:
- $\alpha_n^{g_k}(b) = \alpha_n(b)$ *si* $\phi_n^k(b) = \phi_n^k(\alpha_n(b))$
- $\alpha_n^{g_k}(b) = b$ *sinon.*

La figure 3.10 montre un exemple simple de carte généralisée de dimension 2 multipartitionnée. Notons que notre représentation prend en compte le cas (non illustré sur la figure) où deux liaisons $\alpha_n^{g_i}$ sont superposées.

Les involutions groupantes permettent de définir de nouvelles orbites que nous appelons *orbites de groupe*, auxquelles des informations peuvent être associées (par exemple pour caractériser le matériau de certaines parties d'un bâtiment, e.g. les murs). Notons aussi que, du fait de la définition des fonctions de partition, il est possible de prouver facilement que si α_n^g est une involution groupante définie sur la G-carte $G = (B, \alpha_0, ..., \alpha_n)$, alors $G^g = (B, \alpha_0, ..., \alpha_{n-1}, \alpha_n^g)$ est une G-carte.

3.3.2 Hiérarchie explicite

Nous avons vu précédemment qu'une hiérarchie de cartes généralisées était une multipartition particulière. Les liaisons groupantes peuvent donc aussi, sous réserve de respecter leurs contraintes de définition, servir à optimiser la représentation d'une hiérarchie d'étiquetages puisqu'une hiérarchie, comme nous l'avons dit précédemment, est une multipartition particulière. Néanmoins, bien que cette technique permette de coder plusieurs décompositions avec un coût mémoire raisonnable, elle ne résout pas les deux problèmes suivants :

– le coût de l'évaluation d'un niveau de la hiérarchie ;
– la place mémoire nécessaire pour la représentation d'un bâtiment de plusieurs millions de polygones.

Nous avons donc choisi d'expliciter les niveaux de la hiérarchie afin de limiter les coûts d'évaluation, en représentant la hiérarchie de détails par un "arbre" de cartes généralisées. La racine de l'arbre correspond au détail le plus simple, et chacun des autres niveaux précise la structure de certaines parties du niveau qui lui est supérieur (voir figure 3.11). Notre modèle permet ainsi de travailler localement sur une partie de l'objet en ne traitant qu'une partie de la hiérarchie grâce à une technique de chargement partiel des données en mémoire. Nous échangeons les données entre la mémoire vive et le disque dur pour diminuer le nombre d'informations résidentes au même moment (technique de *swap*). Notons toutefois que ceci impose une définition précise de la relation de filiation et des contraintes de cohérence entre niveaux.

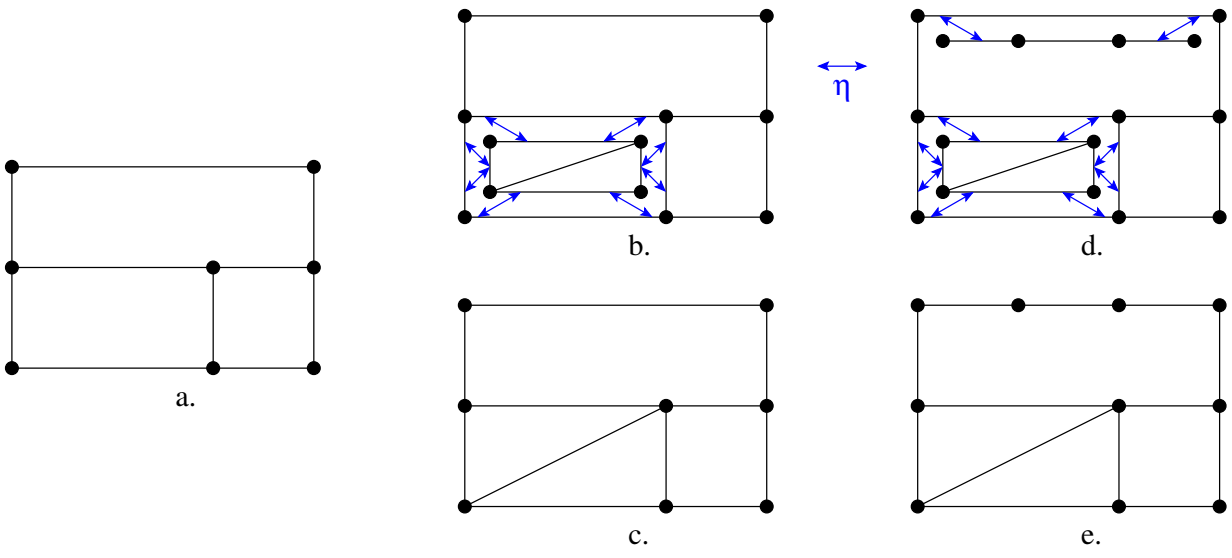


Figure 3.11 – *a. Objet initial correspondant à la racine de la hiérarchie. - b. Une des faces est détaillée et ce détail est relié par la liaison hiérarchique η. - c. L'objet correspondant à la hiérarchie du b. - d. Une des arêtes de la hiérarchie du b est détaillée. - e. L'objet correspondant à la hiérarchie du d.*

Une manière simple d'expliquer ces notions consiste à considérer une hiérarchie de cartes généralisées comme des simplifications successives d'une G-carte initiale (c'est-à-dire le mode

de construction inverse de celui que nous utilisons), par applications de suppression de cellules (voir section 3.2.2).

Étant données deux G-cartes G et G', telles que G' est déduite de G par suppression de certaines cellules, il est possible d'associer de manière unique les brins de G' avec ceux de G n'appartenant pas à des cellules supprimées (voir figures 3.12).

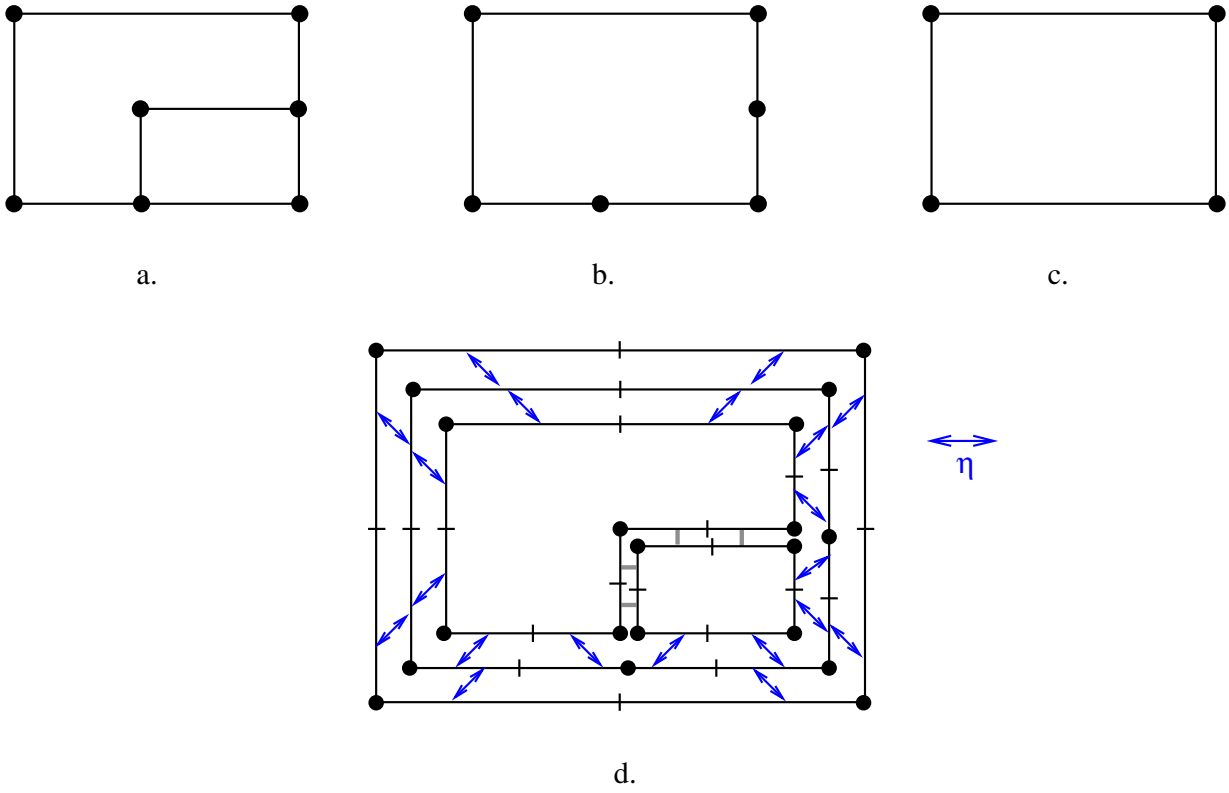


Figure 3.12 – *a. Objet détaillé composé de deux faces. - b. Objet résultant de la succession de deux suppressions de cellules de l'objet a : suppression d'un sommet de degré 2 suivie de la suppression d'une arête. - c. Objet résultant de la suppression de deux sommets de degré 2 de l'objet b. - d. Mise en relation des brins de chaque G-carte pour former une hiérarchie.*

Cette "association" entre les brins des G-cartes des différents niveaux est précisément la relation de filiation entre niveaux. Plus formellement, c'est une injection entre brins d'un niveau et ceux du niveau directement inférieur. Les contraintes de cohérence sur cette injection se déduisent de la définition de l'opération de suppression de cellules. La définition 8 décrit l'ensemble des brins d'une G-carte, dont des cellules ont été supprimées, comme un sous-ensemble des brins de la cellule initiale, nous pouvons donc dire que cette injection est simplement l'application identité. Par la suite, cette injection est notée η, quel que soit le niveau.

Réciproquement, comme nous voulons construire ici un objet en partant du niveau le simple pour aller au niveau le plus détaillé, nous nous basons sur la définition de l'opération inverse de la suppression de cellules : l'opération d'ajout de cellules. Une version très générale de cette opéra-

tion a été proposée dans [Elt94], sous le nom d'opération d'éclatement de cellules. Par exemple, la G-carte de la figure 3.12.b (respectivement 3.12.a) est obtenue par insertion de sommets, ou éclatement d'arêtes (respectivement insertion d'arêtes, ou éclatement de faces) à partir de la G-carte de la figure 3.12.c (respectivement 3.12.b). Cette opération d'éclatement est ici contrainte pour correspondre exactement à l'inverse d'une suite d'opérations de suppression de cellules, telle que définie en définition 8.

Pour réduire la redondance d'informations pouvant exister entre deux niveaux consécutifs de la hiérarchie, nous pouvons remarquer que :

- certaines cellules peuvent ne pas être détaillées d'un niveau à l'autre de la hiérarchie. Par exemple, il est utile de détailler l'intérieur d'une pièce (disposition des meubles), mais l'intérieur d'un mur n'a pas forcément d'intérêt. Dans ce cas, il est inutile de représenter explicitement le mur au niveau plus détaillé de la hiérarchie (voir figure 3.13.a) ;
- si nous souhaitons détailler par exemple la surface d'un mur pour une pièce, il est inutile de dupliquer toute la pièce. Mais dans ce cas, seule la surface du mur pourra continuer à être détaillée (voir figure 3.13.b).

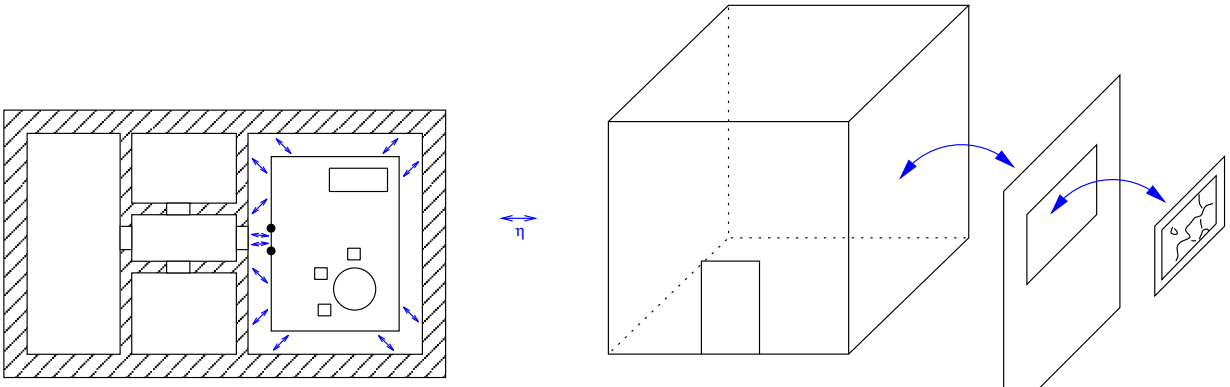


Figure 3.13 – *a. Dans un bâtiment, une pièce est entièrement détaillée : meublée avec une armoire, une table et des chaises. Le détail de la pièce est relié à son niveau supérieur par la liaison hiérarchique η. - b. Ici, seul un des murs de la pièce est détaillé. Tout d'abord, une simple face est placée sur le mur pour représenter un tableau. Puis ce tableau est détaillé par son cadre et sa toile.*

Comme les opérations d'éclatement sont paramétrées par les cellules auxquelles elles s'appliquent et par leur dimension, nous avons pu optimiser la représentation d'une hiérarchie de G-cartes en prenant en compte les remarques précédentes. Néanmoins, cette optimisation en espace s'accompagne d'un certain surcoût en temps puisqu'il est nécessaire de parcourir éventuellement plusieurs niveaux de la représentation optimisée afin de reconstituer un niveau de la représentation explicite.

3.3.3 Le modèle final

Une hiérarchie de G-cartes est donc définie comme une suite de G-cartes éventuellement non connexes, décrivant les détails ajoutés aux différents niveaux. Chaque G-carte est obtenue par l'application d'une suite d'opérations d'éclatement à des cellules de la G-Carte de niveau supérieur dans la hiérarchie. L'application η représente explicitement les relations entre les brins d'une cellule et ceux des cellules résultant des éclatements. De plus, les cellules des G-cartes peuvent être structurées en utilisant le mécanisme de multipartition et en particulier celui des involutions groupantes. Notre modèle a fait l'objet d'un article à l'AFIG [FML02], présentant l'ensemble du travail décrit dans ce chapitre, et à une soumission à Computer Graphics Forum [FML].

Cette structure généralise donc la structure à deux niveaux proposée par Levy [Lev99]. Elle permet de garantir une cohérence topologique entre les niveaux, contrairement aux LOD classiques et à la structure proposée par Guilbert. En revanche, elle ne permet pas d'inclure directement une notion de clonage (voir cependant le chapitre 4, où nous précisons comment l'inclusion de meubles dans une pièce est réalisée à l'aide de ce type de mécanisme). Dans le principe, notre structure se rapproche de la notion de graphe orienté acyclique de simplexes proposé par De Floriani *et al* [DPM97], puisque chaque niveau est défini par la modification du niveau précédent. Néanmoins, notre structure présente l'avantage de fournir une représentation explicite de la scène, tout en respectant les impératifs de faible occupation mémoire et de rapidité d'accès aux informations.

Dans le chapitre suivant, nous présentons le prototype de modeleur de bâtiments que nous avons réalisé. Nous détaillons les principales opérations de manipulation de notre structure, ainsi que les opérations plus élaborées dédiées à la construction de bâtiments. Nous expliquons aussi certains points techniques de l'implantation de ce modeleur (structure hiérarchique, opérations) à partir du noyau de modeleur géométrique basé sur les 3-G-cartes, développé au laboratoire SIC.

CHAPITRE 4

LE MODELEUR DE BÂTIMENTS

Afin de valider cette structure, nous avons développé un prototype de modeleur de complexes architecturaux. Nous avons conçu tout d'abord des opérations de base et élaborées (voir sections 4.1, 4.2 et 4.3) pour construire la structure présentée dans le chapitre précédent. L'implantation de l'ensemble de ces opérations repose sur le noyau topologique développé au sein du laboratoire SIC, permettant de manipuler des 3-G-carte. Nous avons également programmé des opérations spécifiques aux grands bâtiments (section 4.4) et une interface graphique pour le modeleur. Notre prototype est développé sous Windows en C++ avec une interface graphique créée avec Borland C++ Builder.

Nous souhaitons que la construction d'un bâtiment soit la plus intuitive possible. Pour cette raison, nous proposons une modélisation descendante : du détail le plus simple au plus précis. Au niveau de l'interface graphique, nous fixons la profondeur de la hiérarchie à 5 pour un bâtiment : une façade, des murs extérieurs, les étages avec les cloisons intérieures, l'ameublement de chaque pièce sous forme de boîtes englobantes et enfin les meubles.

L'objectif de ce modeleur est de permettre de créer des bâtiments complexes, en précisant le maximum d'informations topologiques, sémantiques et photométriques. Les informations de modélisation permettent de développer des algorithmes adaptés à la visualisation de ces environnements, détaillés dans les deux chapitres suivants. Ce modeleur s'appuyant sur un noyau à base topologique très complet, nous devons utiliser au mieux ses capacités.

4.1 Évolution du noyau topologique

Le noyau de cartes généralisées développé au laboratoire permet de construire des 3-G-cartes, et il possède de très nombreuses opérations utiles en modélisation géométrique par exemple :

- création d'objets de base (parallélépipède, sphère, cylindre, cône, etc) ;
- coutures et décousures de cellules ;
- fusion et contraction de cellules ;
- opérations de triangulations, insertions de cellules ou fermetures de bords ;
- extrusions et révolutions ;
- chanfreinage de sommets ou d'arêtes ;

– opérations booléennes, i.e. union, intersection et différence d'objets.

Ce noyau a été utilisé comme base de développement pour plusieurs applications :

– la conception et le développement d'un modeleur géométrique généraliste ($Moka$), étendu par différents modules adaptés au domaine d'application de la CAO mécanique ;
– la modélisation du sous-sol (calcul de couches géologiques à partir des surfaces d'horizon et de failles, calcul de maillage) ;
– l'analyse d'images, par la conception de modules permettant de calculer des structurations d'images et intégrant aussi des modèles et opérations issues de la géométrie analytique discrète.

Nous avons choisi ce noyau topologique comme base de développement de notre modeleur de bâtiments, et nous avons développé notre structure hiérarchique à partir de l'implantation des 3-G-cartes proposée dans ce noyau.

Il faut noter qu'une des opérations élémentaires les plus utilisées est le parcours d'orbites (par exemple le parcours de l'ensemble des brins composant une cellule à partir d'un brin donné). Ces parcours sont à la base de toute opération de modification ou de visualisation de l'objet. Lorsque les objets sont complexes et que le nombre de brins manipulés devient important, les parcours d'orbites deviennent de plus en plus lents (leur complexité étant fonction du nombre de brins parcourus). Ceci n'est pas gênant pour les opérations de construction, mais pour la visualisation du bâtiment, l'interactivité est indispensable. Même si le nombre de polygones augmente considérablement, un taux de rafraîchissement minimum doit être assuré.

Un parcours d'orbite s'effectue de manière similaire à un parcours de graphe, où les nœuds correspondent aux brins et les arêtes correspondent aux liaisons entre brins par une involution α_i. Ce parcours s'accompagne donc d'un marquage des brins parcourus, et une étape de démarquage est nécessaire à la fin du parcours. La complexité des parcours d'orbite est liée au nombre de brins. Notre décomposition hiérarchique des bâtiments complexes aide donc ici à diminuer le nombre de brins à traiter en permettant de travailler localement sur des petites parties du bâtiment.

La programmation de la structure hiérarchique ne nécessite pas l'ajout d'un grand nombre d'informations supplémentaires : en effet, nous avons essayé au maximum de diminuer le coût mémoire de la structure de données. La classe permettant de représenter un brin d'une 3-G-carte plongée dans $\mathbb{R}^3$ occupe en mémoire environ 80 octets (60 octets pour la topologie et 20 pour la géométrie). Nous avons enrichi cette classe avec 4 octets supplémentaires pour la multipartition (soit 32 partitions différentes pour chaque niveau de la hiérarchie) et 2 octets sous forme de pointeurs pour la représentation de hiérarchie (brin parent et brin enfant). Le surcoût mémoire est donc relativement faible en comparaison de la possibilité de définir des environnements architecturaux complexes.

Quelques opérations de base modifiées

Comme nous l'avons signalé précédemment, les parcours d'orbites nécessitent un marquage des brins. Pour les parcours utilisant plusieurs niveaux de hiérarchie (par exemple, un parcours d'une cellule et de l'ensemble de ses détails), nous avons implanté des marques hiérarchiques pour permettre de marquer plusieurs cartes généralisées différentes dans un même arbre. Pour cela, un tableau de marques est stocké au niveau de la racine de la hiérarchie : demander une

marque hiérarchique revient à demander une marque à chacun des niveaux. Ces marquages sont indispensables pour les propagations de modification ou les procédures d'affichage.

Signalons aussi que les parcours d'orbites ont été modifiés de deux manières différentes. La première permet des parcours d'orbites hiérarchiques, c'est-à-dire le parcours d'une cellule et de tous ses fils. Ces parcours utilisent les marques hiérarchiques. La seconde modification permet de prendre en compte les parcours de groupes. Les liaisons α étant marquées, de nouvelles orbites sont induites : les orbites de groupes. Parcourir un groupe nécessite de vérifier qu'une involution marquée (révélant le changement de groupe) n'est pas franchie. Nous avons programmé ces nouveaux parcours en ajoutant aux parcours existants des tests supplémentaires.

Enfin, pour éviter des redondances d'informations géométriques, les sommets communs à plusieurs détails ont tous pour plongement un même point de $\mathbb{R}^3$. Nous avons donc ajouté à la classe des plongements une variable "membre" donnant son nombre de référencements (nombre de fois où un attribut est référencé par les cellules). Lorsqu'une cellule est supprimée dans un détail, ses plongements doivent l'être aussi. Mais comme les plongements sommets sont communs à plusieurs niveaux de hiérarchie, il faut vérifier le nombre d'occurrences de ces plongements dans la hiérarchie, pour savoir si le point correspondant peut être supprimé.

Les plongements utilisés

Nous avons défini différents plongements au fur et à mesure du développement du modeleur. Comme nous l'avons dit précédemment, les plongements essentiels sont les coordonnées des sommets dans l'espace. Ces coordonnées nous sont utiles pour tous les calculs géométriques. Nous avons aussi utilisé des plongements permettant d'énumérer l'ensemble des objets inclus dans un volume ou dans une face.

En phase de modélisation, il est important de pouvoir nommer les pièces ou les meubles afin d'ajouter des informations sémantiques liées au bâtiment. Nous avons donc des plongements associés aux volumes, permettant de les nommer.

Pour les algorithmes de visualisation, nous avons ajouté aux faces un plongement photométrique, se limitant actuellement à une couleur RVB. Nous pourrons ultérieurement ajouter toutes les informations pouvant être utiles (des textures par exemple). Nous avons aussi utilisé les plongements pour stocker des précalculs géométriques simples, tels que les normales des faces ou les équations de droite des segments, qui sont des données fréquemment utilisées lors de la phase de visualisation.

Pour finir, les bâtiments modélisés sont aussi utilisés pour des calculs de propagation d'ondes radioélectriques (section 7.1.1). Le modeleur permet de caractériser les matériaux de construction des volumes (murs) ainsi que leurs propriétés électromagnétiques.

4.2 La hiérarchie

La définition de la hiérarchie a été simplifiée pour permettre un accès aux données encore plus rapide. En effet, dans le modèle théorique, les i-cellules communes à deux niveaux de détail successifs ne sont pas dupliquées. Pour un volume dont seulement deux faces sont détaillées, nous

avons alors à un niveau donné le volume en question et dans le niveau inférieur les deux faces. En pratique, comme nous l'avons signalé précédemment, les nombreux parcours permettent déjà difficilement de conserver un affichage fluide si l'objet est complexe. Si nous n'exprimons pas explicitement comment sont reliées les faces adjacentes aux faces détaillées, il manque des informations topologiques qu'il faut alors déduire. Ceci complique et ralentit considérablement le parcours des volumes détaillés (parcours des pièces, des étages, etc.).

Nous avons donc choisi de ne supprimer dans la hiérarchie que les n-cellules parmi les i-cellules non détaillées, c'est-à-dire de ne supprimer que les volumes inchangés. Certaines i-cellules redondantes sont donc dupliquées si elles appartiennent à une cellule détaillée, afin de conserver un maximum d'informations topologiques évitant des parcours coûteux.

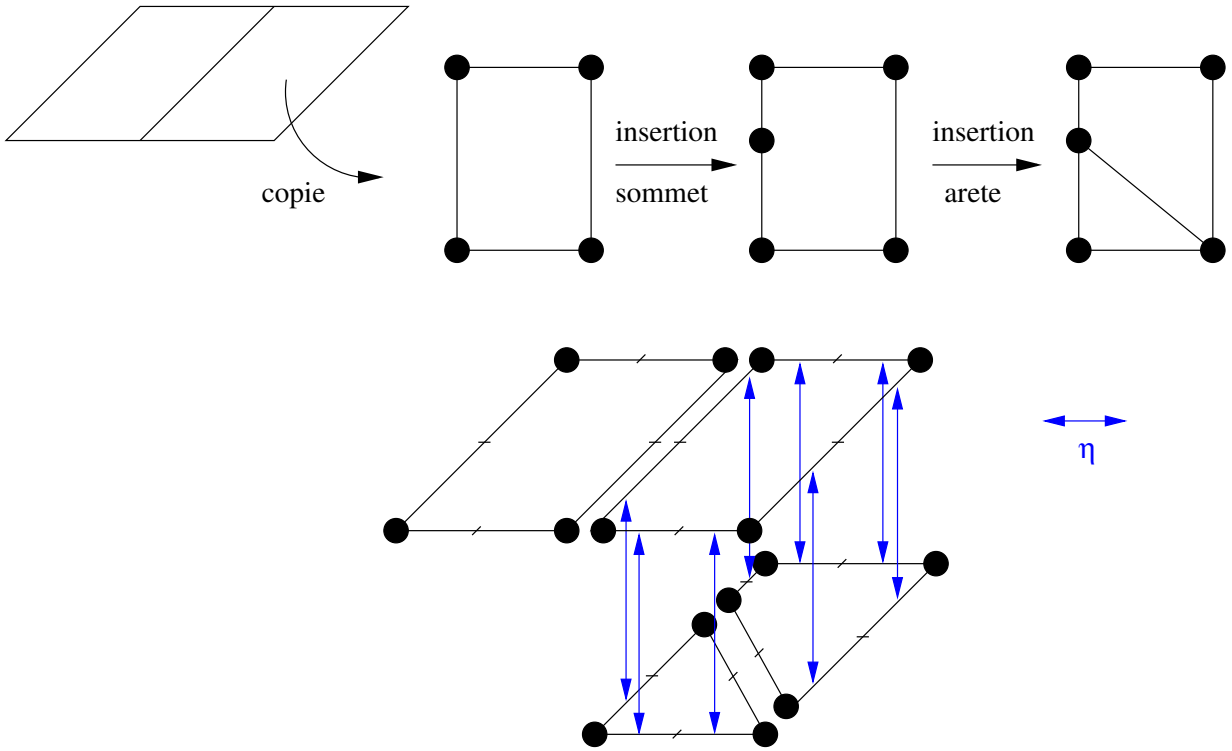


Figure 4.1 – *Le haut de la figure représente le processus permettant de détailler une cellule. Le bas montre la hiérarchie de cartes généralisées obtenue.*

Pour faciliter la modélisation du bâtiment tant du point de vue utilisateur que d'un point de vue technique, la construction de la hiérarchie s'effectue depuis le niveau le plus simple jusqu'au niveau le plus détaillé. Chaque cellule est détaillée indépendamment et remplacée par un ensemble de cellules de même dimension (par exemple, un étage vide est détaillé par un ensemble de pièces et leurs cloisons). À un niveau donné, pour détailler une cellule (par exemple une pièce dans un étage), celle-ci est copiée dans le niveau inférieur, et des opérations topologiques peuvent lui être successivement appliquées (par exemple des insertions de i-cellules, des extrusions, etc). Ces opérations ne doivent pas modifier le profil de la cellule et ne changent donc que l'intérieur de celle-ci. De cette manière, nous pouvons définir au moment de la copie un lien η entre les deux descriptions de la cellule : η associe chaque brin à sa copie (figure 4.1).

Les opérations élémentaires de manipulation de la hiérarchie sont les suivantes :
(i) $createDetail(b)$ copie dans le niveau inférieur de la hiérarchie les brins appartenant à la n-cellule du brin b et les lie à leur copie à l'aide de la liaison η.
(ii) $findParent(b)$ renvoie la cellule parent du brin b. Notez que b peut ne pas être directement lié par η à un brin de la cellule parent (rappelons que tous les brins parents sont reliés par η à un brin fils, mais l'inverse n'est pas vrai). Pendant le processus de modélisation, cette opération est employée uniquement quand il est nécessaire de propager les résultats d'une modification à un niveau supérieur de la hiérarchie (par exemple lors de la création d'une ouverture sur l'extérieur).

De la même manière, le dernier niveau de détail contient les pièces les plus détaillées (pièces meublées) et possède donc un très grand nombre de polygones. Après avoir modélisé quelques bâtiments simples (moins de $500\,000$ faces), nous avons remarqué que les meubles représentaient plus de 90% de la totalité des faces du bâtiment. Afin de pouvoir modéliser de plus grands bâtiments, les meubles sont représentés au niveau des pièces par leurs boîtes englobantes, et le dernier niveau de la hiérarchie contenant leur détail n'est pas stocké entièrement en mémoire. Les meubles sont stockés sur le disque dur et seuls les meubles de la pièce courante sont présents en mémoire si l'utilisateur le demande.

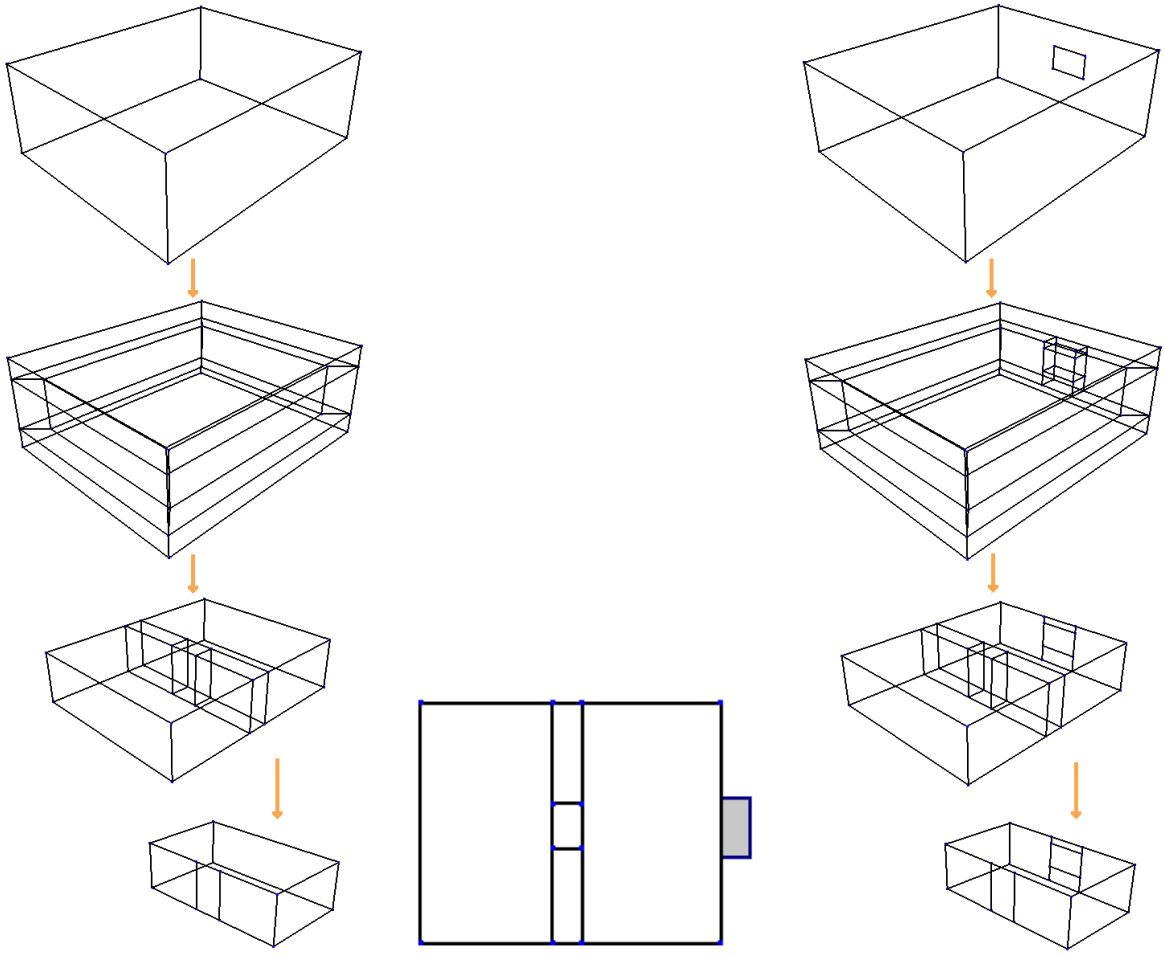
Certaines opérations de haut niveau faisant partie du noyau (extrusions, insertions des cellules, etc.) ont été étendues à la structure hiérarchique pour permettre de modifier plusieurs niveaux de hiérarchie en même temps.

Prenons l'exemple de l'insertion d'une fenêtre dans un mur. Cette opération, au niveau des étages, correspond à l'insertion d'un volume à l'intérieur du mur à la position indiquée par l'utilisateur. Néanmoins, une fenêtre donnant sur l'extérieur doit être présente à chaque niveau de la hiérarchie (sauf celui des meubles) : la façade extérieure, les murs principaux du bâtiment, l'étage correspondant et la pièce sur laquelle elle donne. Nous devons insérer plusieurs cellules à différents niveaux. Sur la figure 4.2, en comparant la hiérarchie avant insertion (à gauche) à celle après insertion (à droite), nous pouvons voir ces différentes cellules insérées : une face au niveau de la façade du bâtiment, un volume dans un des murs porteurs, une face au niveau de l'étage et une au niveau du détail de la pièce. Au centre de la figure, le plan de l'étage est représenté avec en évidence la fenêtre sur l'extérieur qui a été insérée.

4.3 La multipartition

Une involution groupante α_n^g est représentée par un booléen associé au pointeur symbolisant l'involution correspondante α_n. Ce booléen indique si les deux n-cellules connexes appartiennent au même groupe (voir le premier schéma de la figure 3.10). Un octet permet donc de représenter simultanément 8 partitions différentes. Cette représentation nous semble un bon compromis permettant à la fois de réduire la taille de stockage et le temps d'accès aux données pour parcourir toutes les cellules d'un même groupe.

Des opérations de base permettent de construire et modifier les multipartitions :
(i) $linkGroup(b_1, b_2, g)$ lie b_1 et b_2 par α_n^g : cette opération est effectuée seulement si les deux brins b_1 et b_2 sont déjà liés par α_n. Elle ne modifie que la marque booléenne correspondante.
(ii) $sewGroup(b_1, b_2, g)$ coud la $(n-1)$-cellule incidente à b_1 à celle incidente à b_2 par α_n^g.



a. b. c.

Figure 4.2 – *À gauche, une vue 3D des différents niveaux d'un bâtiment simple. Au centre, une vue 2D de l'étage indiquant l'emplacement de la fenêtre y étant ajoutée. À droite, le résultat de l'opération d'insertion d'une fenêtre sur les différents niveaux.*

Cette opération consiste à parcourir l'orbite $< \alpha_0, ..., \alpha_{n-1} > (b_1)$ pour appliquer sur chaque brin l'opération de liaison $linkGroup$. Elle vérifie aussi l'unicité des plongements de groupe : si parmi les anciens groupes associés à b_1 et b_2, chacun possédait un attribut du même type, alors celui du groupe associé à b_2 est supprimé.
(iii) $addAttribute(b, g, Att)$ ajoute un plongement (ou attribut) Att (une coordonnée de sommet, une couleur de face, etc.) au groupe de numéro g associé à b. Avant de placer cet attribut, $uniqueGroupAtt$ permet de vérifier qu'aucun attribut de même type ne soit déjà présent. Dans ce cas, l'attribut Att est associé à un brin du groupe, sinon l'attribut déjà existant est remplacé par Att.
(iv) $uniqueGroupAtt(b, g, Att)$ vérifie que le groupe g associé au brin b ne contient pas déjà le

même type d'attribut que Att. Elle renvoie l'attribut en conflit sinon.
(v) $getAttribute(b, g, Type)$ recherche un attribut de type $Type$ se trouvant sur le groupe g associé au brin b.
(vi) $isSameGroup(b_1, b_2, g)$ renvoie vrai si b_1 et b_2 appartiennent au même groupe. Pour cette opération, tous les brins du groupe doivent potentiellement être parcourus. Cette fonction est employée uniquement par $unsewGroup$. C'est le seul cas où nous avons besoin de tester si deux brins appartiennent au même groupe.
(vii) $unsewGroup(b_1, b_2, g)$ annule le groupement de deux brins avec pour résultat : $\alpha_n^g(b_1) = b_1$ et $\alpha_n^g(b_2) = b_2$. Dans la pratique, ceci revient à inverser la marque booléenne associée à b_1 et b_2. Nous devons également vérifier si d'autres brins du groupe sont encore reliés. Les attributs communs aux cellules du groupe sont reproduits dans chaque sous-groupe nouvellement créé.

4.4 Étapes de modélisation d'un bâtiment

Notre modeleur permet de décrire un grand bâtiment progressivement en commençant par son profil extérieur pour ensuite détailler étape par étape son intérieur. L'utilisateur peut travailler avec des vues fil de fer ou polygonale (voir figure 4.3). Il peut utiliser un ensemble d'opérations de construction que nous détaillons dans les sous-sections suivantes : création d'un profil, disposition des murs, des ouvertures, etc. Toutes ces opérations ont été développées pour manipuler une subdivision de l'espace $\mathbb{R}^3$ afin de construire un environnement aussi réaliste que possible. Chaque objet modélisé est représenté par une 3-cellule ou un groupe de 3-cellules connexes. Par exemple dans un bâtiment, les murs (3-cellules) délimitent des pièces (3-cellules), connectées par des ouvertures (également des 3-cellules). Dans une partition, les groupes sont constitués de volumes (pièces, murs, ouvertures) cousus par α_3.

Le bâtiment est décomposé en 5 niveaux de hiérarchie. Le premier niveau représente la façade du bâtiment : ce niveau très simplifié représente le contour du bâtiment et l'emplacement de ses ouvertures donnant sur l'extérieur. Le deuxième niveau décrit les murs porteurs ainsi que les sols et plafonds des différents étages. Le troisième niveau montre l'agencement des pièces à l'intérieur de chaque étage, les cloisons les séparant et les ouvertures qui les relient. Le quatrième niveau décrit l'ameublement de chaque pièce sans description des meubles : seules les boîtes englobantes des meubles sont représentées, et chacune d'elles possède une référence sur l'objet qu'il contient (nom de fichier). Le dernier niveau de la hiérarchie n'est pas complet. Il décrit les meubles eux-mêmes, mais ces derniers étant trop nombreux et trop détaillés pour tenir tous en mémoire, le niveau ne contient que les meubles de la pièce courante et seulement si l'utilisateur le demande.

Lors de la modélisation, les structures de hiérarchie et de multipartition sont masquées à l'utilisateur. Ce dernier n'a pas besoin de connaître la structure sous-jacente au modeleur. La hiérarchie est créée automatiquement par le modeleur. L'utilisateur a néanmoins à sa disposition un composant lui permettant de naviguer dans cette hiérarchie : un arbre de navigation visible à gauche de l'interface (figure 4.3). Au centre, l'utilisateur peut voir la partie du bâtiment qu'il a sélectionnée, avec tout autour un ensemble de boutons permettant de modifier l'affichage et de

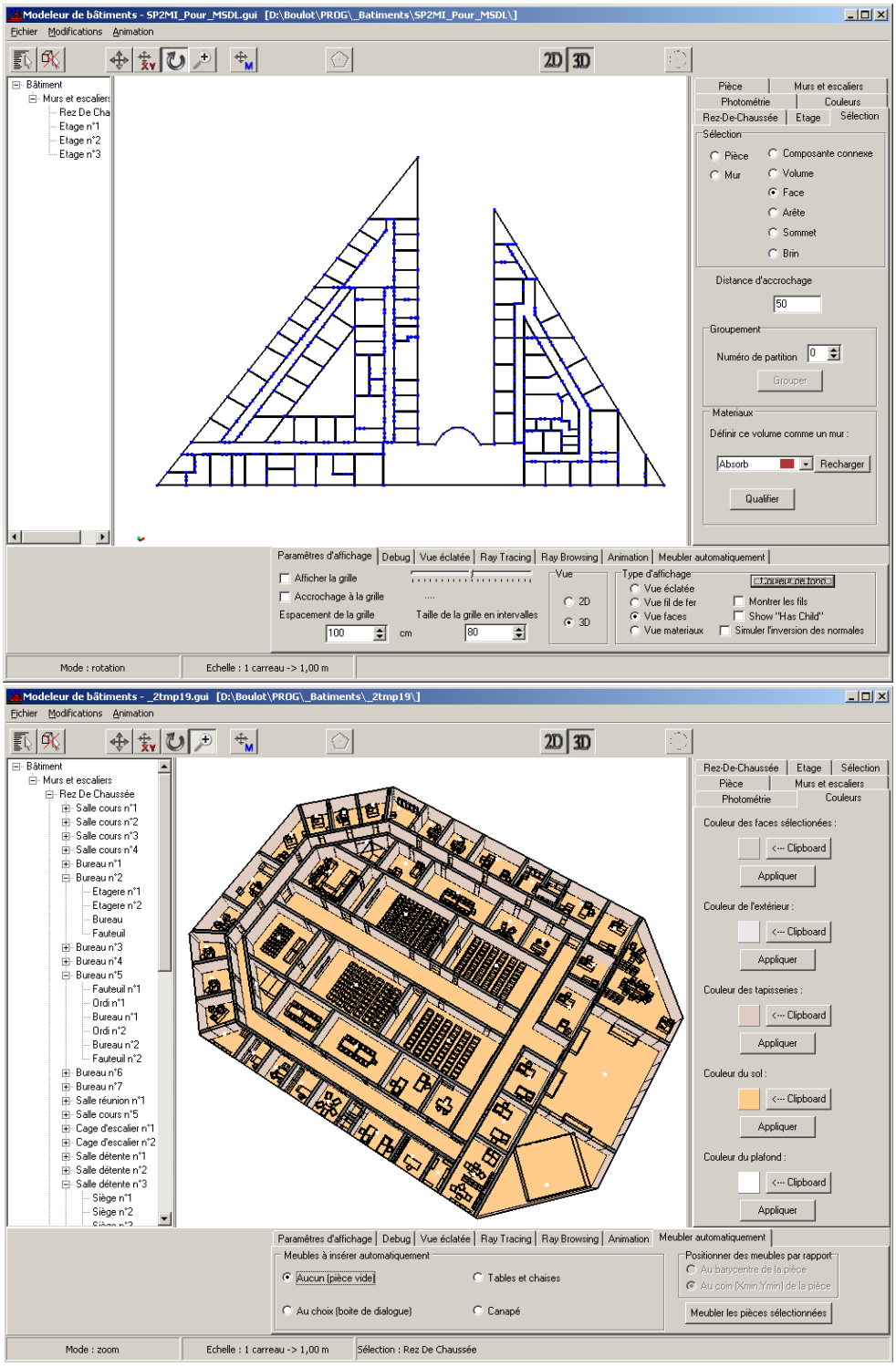


Figure 4.3 – *Images de l'interface du modeleur : en haut, l'étage d'un immeuble non meublé, et en bas, celui d'un bâtiment entièrement meublé.*

construire le bâtiment à l'aide des opérations mises à sa disposition. Nous décrivons dans la suite les différentes étapes de modélisation.

Structure du bâtiment

L'utilisateur crée un polygone représentant le profil extérieur du bâtiment (figure 4.4.a) puis saisit le nombre d'étages et l'épaisseur des murs extérieurs. Ce polygone est extrudé pour créer un volume correspondant à la façade du bâtiment (figure 4.4.c). Cette façade est stockée dans la G-Carte racine de la hiérarchie.

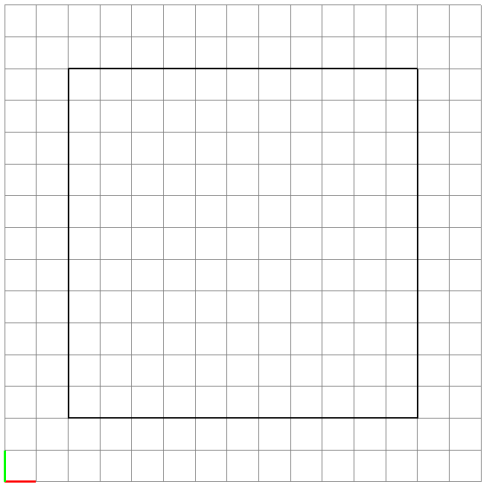
Un second niveau de hiérarchie est alors créé et une suite d'extrusions est appliquée à une copie du polygone dessiné pour définir les murs externes (figure 4.4.b) et chaque étage (figure 4.4.d). Les deux niveaux de hiérarchie sont automatiquement liés : chaque brin du premier niveau est lié à son brin fils à l'aide d'une liaison η.

Édition des étages

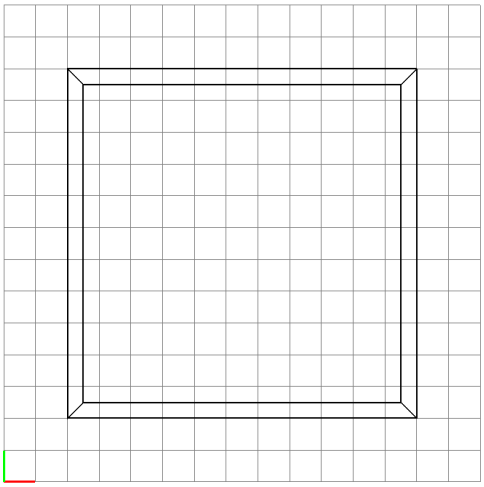
Quand l'utilisateur souhaite détailler un étage, le volume correspondant est copié dans le troisième niveau de hiérarchie et les liaisons η sont automatiquement créées. Notez que chaque étage peut être indépendamment édité par l'utilisateur. Le détail d'un étage déjà édité peut à tout moment être reproduit dans un autre.

L'utilisateur peut décrire dans l'étage sélectionné les cloisons intérieures. Sur une vue 2D, une ligne polygonale est tracée et extrudée pour définir le mur (figure 4.5.e). Quand le mur décrit est en contact avec un autre mur ou une paroi de l'étage, une couture par α_3 fait suite à l'insertion d'une nouvelle face dans le mur intersecté, de sorte que la cohérence topologique soit préservée. Les murs intérieurs sont groupés tous ensemble de manière transparente pour l'utilisateur ; ils forment ainsi une partition de l'étage. Notons que la création d'un mur divise le volume de l'étage en trois nouveaux volumes : le mur et les deux pièces résultantes. La description des murs intérieurs permet donc de déduire automatiquement chacune des pièces.

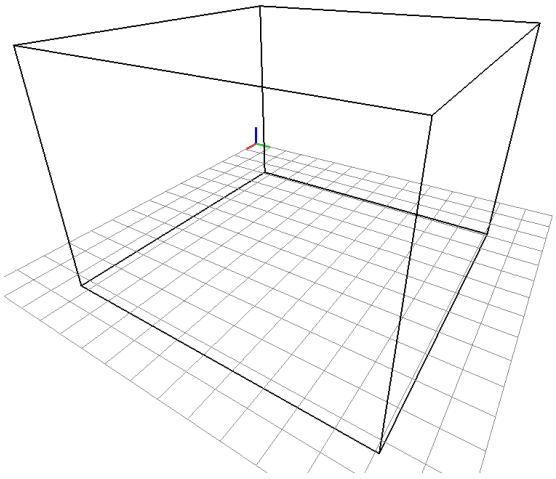
Plusieurs types d'ouvertures sont proposés à l'utilisateur : des portes, des fenêtres ou des baies. Elles peuvent être placées sur les murs intérieurs ou extérieurs (figure 4.5.f). Dans le cas des murs externes, des modifications topologiques doivent être propagées du niveau courant à la racine de hiérarchie. Un ensemble d'opérations spécifiques de plus haut niveau a été conçu pour mettre à jour la hiérarchie du bâtiment. Une ouverture correspond à un volume vide inséré à l'intérieur d'un mur ; une face est insérée de chaque côté de mur et le volume est placé entre ces deux faces de manière à préserver la topologie (un ensemble de faces supplémentaires doit être ajouté en même temps que le volume). La figure 4.6 illustre le résultat de ces deux opérations. Dans le cas des ouvertures donnant sur l'extérieur, les faces et les volumes sont insérés à plusieurs niveaux de la hiérarchie. Les faces (2-cellules) partagées par une ouverture et une pièce sont considérées comme transparentes (association d'attributs photométriques *ad hoc*). De plus, les pièces et leurs ouvertures sont groupées à l'aide du même α_3^g que les murs. Nous avons ainsi une partition complète du bâtiment en deux groupes : les pièces et les murs. Notons ici que ces deux groupes sont importants, puisque le groupe des pièces est utilisé pour la visualisation et



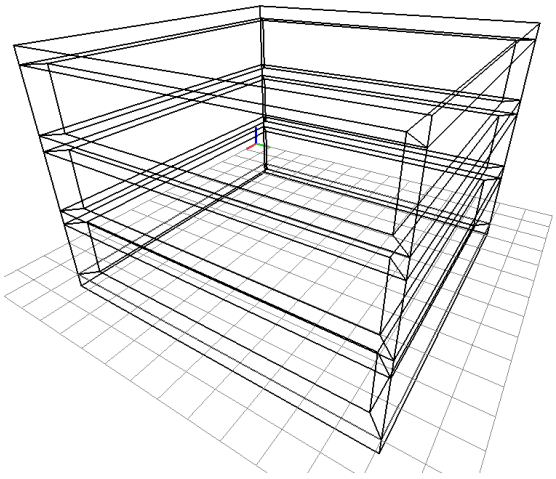
a. Contours du bâtiment déssiné par l'utilisateur.



b. Première extrusion pour former les murs intérieurs.



c. Extrusion du contour *(a)* pour obtenir le profil extérieur du bâtiment.

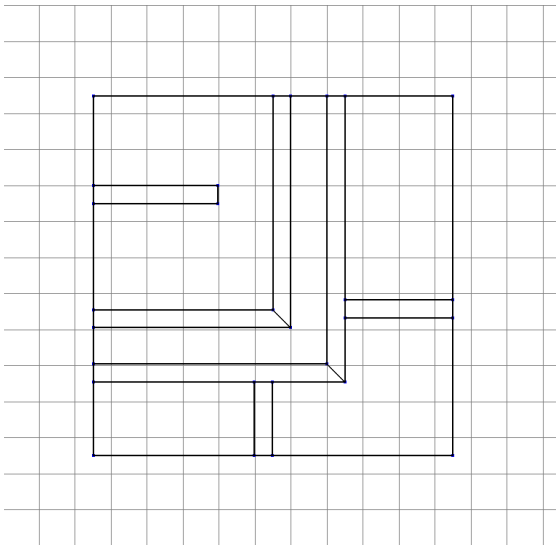


d. Extrusions de la forme géométrique *(b)* pour créer chaque étage.

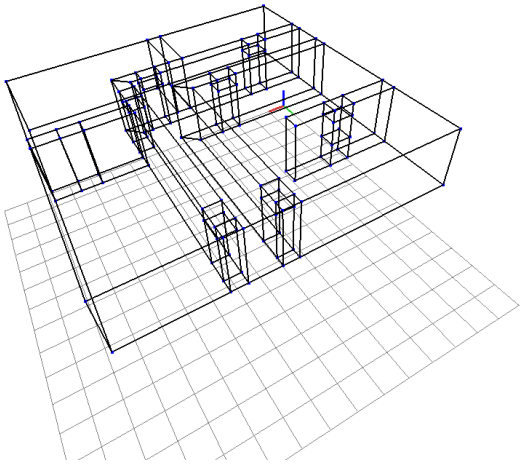
Figure 4.4 – *La structure du bâtiment est générée à l'aide d'une série d'extrusions à partir des données de l'utilisateur : la forme de l'emprise au sol du bâtiment, le nombre d'étages et l'épaisseur des murs extérieurs.*

la simulation d'éclairage, et le groupe des murs pour les simulations de propagation d'ondes électromagnétiques.

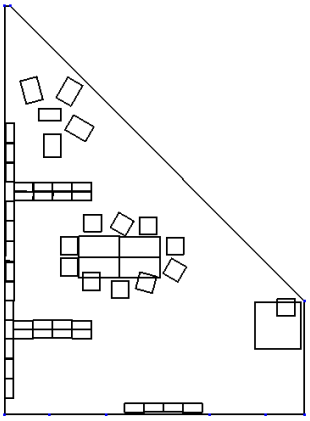
Pendant la création des murs et des ouvertures, des attributs spécifiques sont automatiquement associés à chaque volume pour leur donner une sémantique : les murs, les salles, les portes



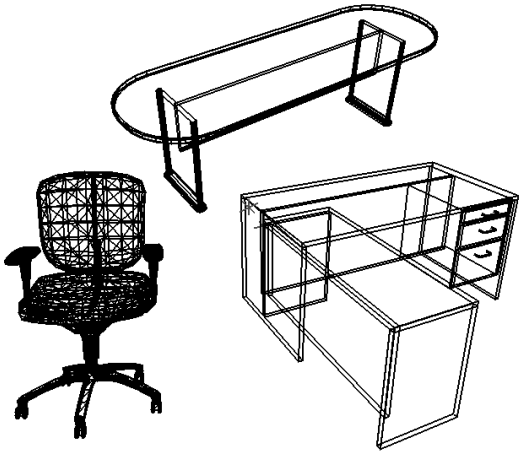
e. Disposition des murs/cloisons d'un étage.



f. Mise en place des ouvertures.



g. Des boîtes englobantes pour représenter les meubles au 4^{e} niveau de la hiérarchie.



h. Exemples de meubles.

Figure 4.5 – *Étapes de création des murs intérieurs, des ouvertures et de l'ammeublement.*

et les fenêtres. L'utilisateur a également la possibilité d'ajouter ses propres attributs aux volumes : des matériaux (béton, plâtre, etc.) et des informations sémantiques (bureaux, bibliothèque, toilettes, etc.). Ceci peut aussi se faire pour les faces en associant des attributs photométriques tels que la couleur du papier peint, les textures ou un nom de BRDF. Des opérations de plus haut niveau sont également employées pour grouper des pièces adjacentes partageant des ouvertures, en utilisant les opérations de gestion des multipartitions.

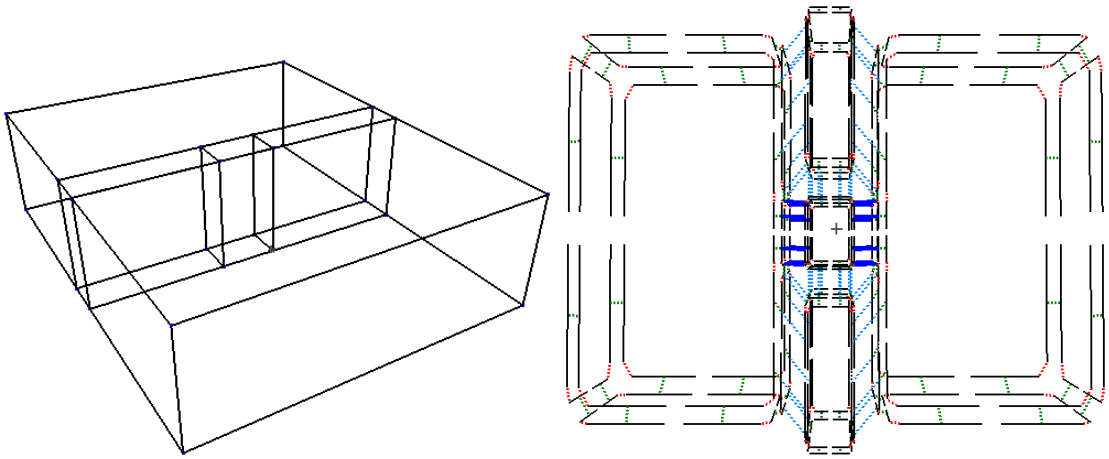


Figure 4.6 – *Une ouverture reliant deux pièces : à gauche, vue fil de fer des deux pièces, à droite, vue éclatée de la carte généralisée montrant les liaisons groupantes en gras.*

Ameublement des pièces

Chaque pièce peut être décrite au 4^{e} niveau de la hiérarchie. Nous pouvons employer des meubles décrits à l'aide du format de sauvegarde du noyau sur lequel repose notre modeleur, ou sous forme de liste de polygones. Au niveau de la pièce, les meubles sont représentés par leur boîte englobante et le dernier niveau de hiérarchie contient les détails des objets.

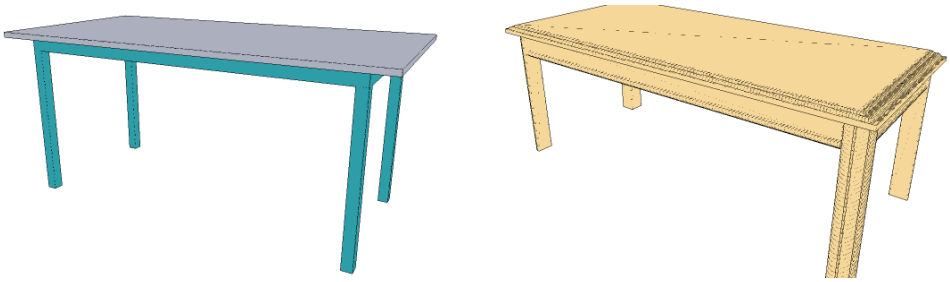


Figure 4.7 – *À gauche une table simple composée de* 30 *rectangles et à droite une table travaillée composée de* 3970 *polygones.*

En pratique, en raison du nombre élevé de polygones, la géométrie du mobilier et sa topologie sont stockées sur le disque, excepté pour la salle en cours d'édition. En effet, la figure 4.7 montre deux exemples extrêmes d'un même type de meuble : une table modélisée avec seulement 30 rectangles et une table plus travaillée possédant 3970 triangles. En fonction des meubles que nous choisissons pour les pièces, le nombre de polygones nécessaires pour modéliser un bâtiment complet peut devenir critique. Nous utilisons donc le disque dur pour stocker les meubles sur lesquels nous ne travaillons pas directement, afin d'alléger le nombre d'informations résidant en mémoire vive. Les meubles sont donc représentés par leurs boîtes englobantes. Elles contiennent un attribut indiquant le nom du fichier contenant la description de l'objet et les transformations

géométriques (translations, rotations) à appliquer. Le même fichier peut être employé plusieurs fois. Nous simulons ainsi le principe classique de clonage d'objets.

Une fonction d'importation d'objets décrits par des listes de faces permet d'insérer des meubles dans les pièces. Les formats choisis sont les formats VRML 1.0 / Inventor, VRML 2.0 Ascii et NFF. La présence des boîtes englobantes utilisées pour représenter les meubles au niveau des pièces se justifie doublement : premièrement pour des raisons de coût mémoire et de vitesse d'affichage, deuxièmement parce que nous n'avons aucune garantie concernant la cohérence topologique des objets récupérés.

Dans le cas d'objets décrits par une liste de polygones, la topologie peut partiellement être retrouvée. L'objet est d'abord corrigé en supprimant les triangles dégénérés et en triangulant les polygones non coplanaires ; les liaisons α_0 et α_1 sont automatiquement déduites à partir de la liste résultante de polygones. Ce processus très simple consiste à créer chaque polygone indépendamment à partir de la liste de faces. Deux brins sont créés pour chaque arête et sont liés par α_0. Les arêtes correspondantes sont ensuite reliées entre elles au niveau des sommets par α_1. Pour déterminer les liaisons α_2, le principe est de rechercher les faces partageant une même arête. Lorsqu'il en existe deux, il suffit de les coudre par α_2. Néanmoins, les objets ne sont pas toujours des objets topologiquement corrects, encore moins des variétés. Il est même possible que l'objet ne soit pas orientable ou qu'il s'agisse d'une non-variété. Si un nombre pair de faces partagent une même arête, nous les relions deux à deux par α_2 après avoir classé les faces dans l'ordre trigonométrique. Dans le cas contraire, nous préférons ne pas coudre pour conserver la cohérence topologique de l'objet, même si nous perdons ainsi les informations sur les volumes.

Il en est de même pour le calcul de α_3. Lorsque deux surfaces (plusieurs faces cousues par α_2) possèdent deux faces identiques (au sens géométrique), les deux faces sont cousues par α_3. Mais il est rare que nous puissions retrouver des informations sur les volumes composant l'objet. En général, le nombre de faces des objets a été réduit pour permettre d'en accélérer l'affichage. Les faces intérieures des objets n'étant pas visibles, elles sont rarement conservées, ce qui rend très difficile la reconstruction des involutions α_3.

Une fois les meubles importés dans la pièce, l'utilisateur peut les déplacer à l'aide de translations et rotations appliquées uniquement à la boîte englobante afin de conserver un déplacement interactif. Si l'utilisateur le souhaite, il est possible d'afficher les faces de l'objet lors de son déplacement, mais ceci ralentit légèrement le déplacement en raison du plus grand nombre de faces à traiter.

Pour accélérer l'ameublement, la disposition des meubles d'une pièce peut être reproduite dans plusieurs autres pièces du bâtiment. De plus, un mécanisme de script décrivant l'agencement des meubles a été réalisé afin d'ajouter automatiquement des objets à une salle donnée avec de légères modifications aléatoires. La figure 4.8 montre un exemple de script pouvant être utilisé en entrée du module d'ameublement du modeleur. Le format est des plus simples : les trois premières colonnes indiquent les translations suivant l'axe X, Y et Z, la quatrième colonne correspond à une rotation suivant Z et la dernière contient le nom du fichier contenant le meuble à insérer. Les translations et la rotation peuvent être décrites soit par un nombre exact soit par un intervalle (A~B), dans lequel un nombre aléatoire est tiré. Le hasard permet de détailler plusieurs pièces avec le même script sans qu'elles soient trop identiques. Les formats de fichiers décrivant

```
# Tx       Ty       Tz      Rz         meuble
5.5         3         0      90      bureau_angle.nff
5.5       2.7         0   70~100     fauteuil_bureau.fsm
5.75  3.25~3.5    1.05      80      ordinateur.wrl
```

Figure 4.8 – *Exemple de script d'ameublement.*

les meubles sont soit les formats précédemment cités ne donnant que la géométrie de l'objet (NFF, VRML, etc.) soit des fichiers de cartes généralisées (fichiers FSM propres au modeleur).

Bâtiment complet

À tout moment, l'utilisateur peut changer de point de vue dans la hiérarchie grâce à la fenêtre d'arborescence (figure 4.3 sur la gauche), ou afficher la totalité d'un étage (vue du bas de la même figure). Cette arborescence est générée automatiquement lors de la modélisation, mais la sémantique (les noms de chaque étage, pièce ou meuble) peut être changée par l'utilisateur. Par exemple, sur la deuxième image de la figure 4.3, les pièces et les meubles ont tous été nommés.

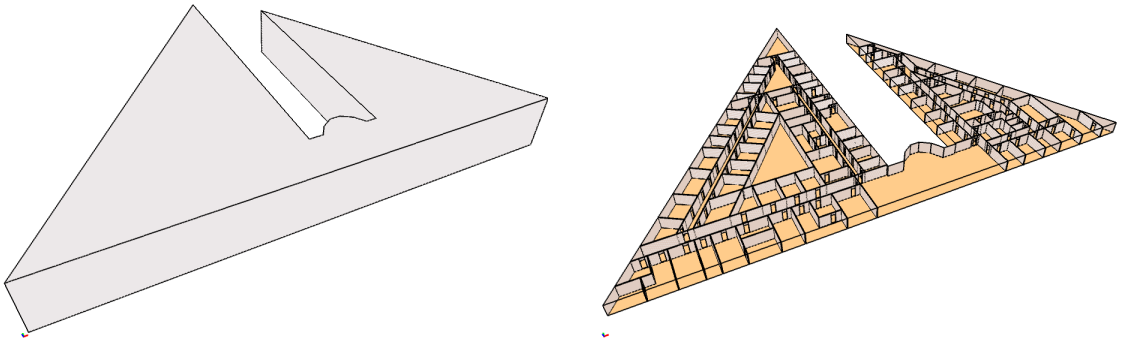


Figure 4.9 – *Bâtiment inspiré d'un des immeubles hébergeant notre laboratoire : le SP2MI.*

Les affichages sont réalisés à l'aide de parcours en profondeur dans la hiérarchie. Les multipartitions peuvent être mises en évidence lors de la sélection explicite d'un groupe. De la même manière, la hiérarchie est visible à l'aide du composant de navigation dans la structure du bâtiment (à gauche sur l'interface). Néanmoins, il est possible de visualiser directement la hiérarchie et la multipartition à l'aide du mode expert : bouton "voir la topologie". Nous obtenons alors des vues comme celle de la figure 4.10 : les niveaux de la hiérarchie sont affichés en mode "éclaté", les liaisons η ou les liaisons groupantes pouvant être aussi représentées. Pour la hiérarchie, l'affichage fil de fer ou polygonal permet aussi à l'aide de l'option "montrer les fils" de visualiser un

sous-arbre, c'est-à-dire tous les niveaux de détail d'une même cellule (sauf les meubles pour des raisons d'espace mémoire). Nous expliquons dans la section suivante comment ces affichages sont réalisés.

Pour finir, nous montrons quelques images issues de notre modeleur. La figure 4.3 nous montre l'interface du modeleur ainsi que deux bâtiments complexes qui ont été construits avec celui-ci. Le bâtiment triangulaire est inspiré d'un des bâtiments hébergeant le laboratoire SIC : le SP2MI sur le site du Futuroscope. Sur la figure 4.9, nous pouvons voir sa façade extérieure et le détail d'un de ses étages. Ce bâtiment n'est pas encore meublé, et ne possède pour l'instant pas de fenêtres donnant sur l'extérieur. Le second bâtiment de la figure 4.3 est un bâtiment imaginaire en forme d'octogone, notre plus grand bâtiment : il est composé de plus de 5 millions de polygones. Il s'agit d'un bâtiment entièrement meublé, seules les fenêtres donnant sur l'extérieur n'ont pas encore été placées. Enfin, la figure 4.11 montre un autre bâtiment aux formes plus complexes. Il possède différents types d'ouvertures (portes, fenêtres) et il est meublé. Le bâtiment le plus complet est le bâtiment en forme de L présenté sur la figure 4.13.

4.5 Visualisation à l'aide d'OpenGL

Pour la phase de modélisation, nous avons besoin d'un affichage :

- fluide, le déplacement des objets ou la visualisation du bâtiment ne doivent pas être saccadés sinon l'utilisateur ne peut pas aisément modéliser les détails ;
- sélectif, la totalité du bâtiment n'est pas affichée en permanence aussi bien pour des raisons techniques (coût mémoire) que pour des raisons de lisibilité de l'information (travail local sur une pièce) ;
- paramétrable, l'utilisateur peut souhaiter afficher les objets en mode fil de fer (pour mieux voir la structure générale), en mode polygonal (pour voir les couleurs des faces), en mode volumique (pour distinguer les différents matériaux de construction) ou encore en mode éclaté (affichage des brins de la G-carte).

L'ensemble de ces contraintes implique qu'il est nécessaire de travailler avec des parcours de i-cellules pour afficher des arêtes (1-cellules), des faces (2-cellules) ou des volumes (3-cellules). Nous devons aussi prendre en compte les informations hiérarchiques afin de pouvoir travailler sur un détail particulier et ses fils. Enfin, la sémantique est aussi importante : par exemple, lorsque nous travaillons sur une pièce, les opérations (ameublement) ne sont pas les mêmes que pour un étage (création de pièces).

Lors de l'étape de modélisation, l'affichage le plus utilisé est le mode fil de fer. Il permet de voir les modifications de la scène, même dans les parties non visibles en mode polygonal. L'affichage des brins est assez rarement utilisé puisqu'il a été principalement programmé pour vérifier la validité de nos opérations et pour des démonstrations de la structure. Un utilisateur modélisant un bâtiment n'a pas besoin de connaître la structure sous-jacente.

Pour les étapes de visualisation et de simulation, nous avons développé les vues polygonale et volumique. La première permet la définition des propriétés photométriques des faces pour la visualisation et la simulation d'éclairage. La seconde permet de caractériser les matériaux de construction pour la simulation de propagation d'ondes.

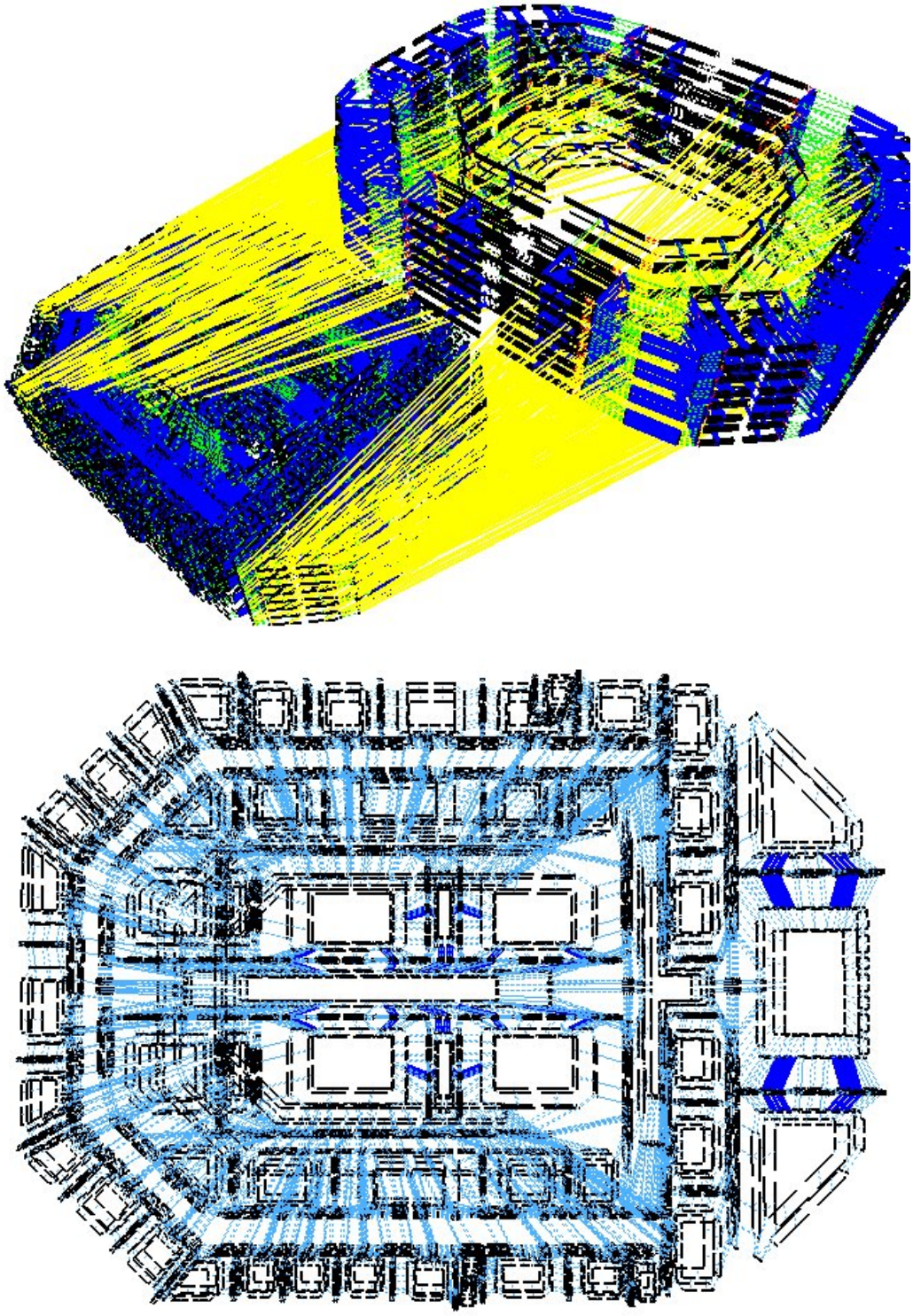


Figure 4.10 – *En haut, affichage éclaté de deux niveaux de la hiérarchie (la disposition des étages et celle des pièces), la liaison hiérarchique η étant mise en évidence. En bas, affichage éclaté du niveau inférieur, certaines liaisons groupantes étant représentées en gras.*

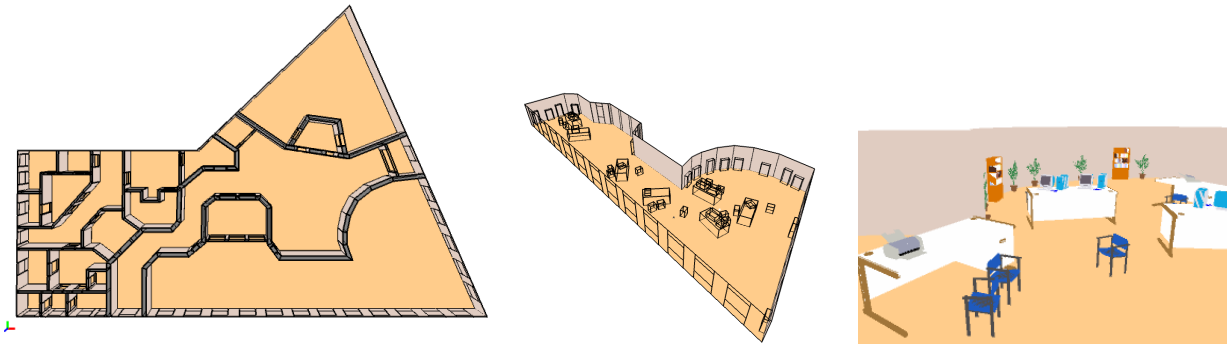


Figure 4.11 – *Bâtiment comprenant des pièces de formes complexes, appelé Zarbi.*

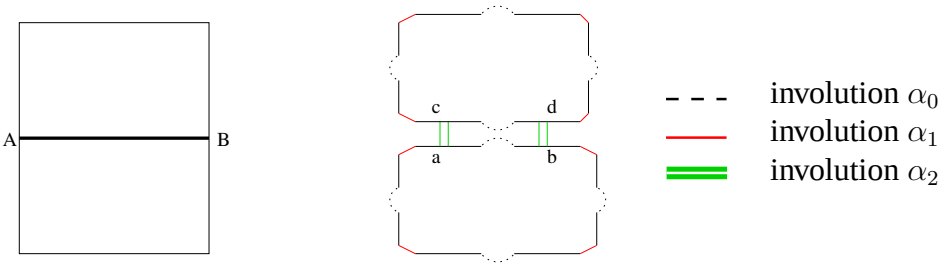


Figure 4.12 – *À gauche, la carte généralisée en affichage fil de fer dont un segment AB est mis en évidence. Á droite, la même carte généralisée affichée en mode éclaté et les brins a, b, c et d correspondant à l'arête AB.*

Pour une carte généralisée, l'ensemble de ces vues peut être obtenu par des parcours classiques d'orbites. Dans le chapitre suivant, nous revenons sur ce problème lors du lancer de rayons. Prenons l'exemple de l'affichage fil de fer, le principe des autres vues étant semblable. Afin d'afficher l'ensemble des arêtes composant un objet, nous devons parcourir toutes les orbites correspondant aux 1-cellules incidentes à chaque brin. Étant donné qu'une arête est composée de plusieurs brins, si nous affichons la 1-cellule pour chaque brin qu'elle contient, l'arête sera affichée plusieurs fois. Un marquage est utilisé pour éviter cette redondance d'affichage. Sur l'exemple de la figure 4.12, pour afficher le segment AB composé ici de quatre brins, nous devons connaître un brin b de ce segment (le choix d'un brin particulier n'a pas d'importance). À partir de b, nous pouvons récupérer les coordonnées du sommet B. Pour connaître le deuxième sommet extrémité du segment, nous récupérons les coordonnées du sommet A associé au brin $a = \alpha_0(b)$. Nous marquons l'ensemble des brins appartenant à la 1-cellule de b, c'est à dire ceux de l'orbite $< \alpha_0, \alpha_2, \alpha_3 > (b) = \{a, b, c, d\}$, pour ne pas redessiner ce segment. En répétant cette opération pour tous les brins non marqués à afficher, nous obtenons la liste des segments de la vue fil de fer. Ce principe est le même pour l'affichage polygonal en utilisant d'autres parcours.

Pour le modèle hiérarchique, le principe d'affichage est similaire, à la différence que nous ajoutons une option permettant d'afficher en plus de la cellule l'ensemble de ses détails. Pour ce faire, nous allouons une marque hiérarchique indiquant l'ensemble des brins du sous-arbre de détail de la cellule. Reprenons l'exemple de l'affichage fil de fer. Nous parcourons toujours l'ensemble des brins de la cellule devant être affichée. Ce parcours nécessite une marque classique pour repérer les brins parcourus. En parallèle au parcours, si nous trouvons un brin possédant un fils (liaison η définie), nous marquons à l'aide de la marque hiérarchique le fils comme devant être dessiné. Une fois les brins de la G-carte affichés, la G-carte fille est parcourue pour afficher les brins marqués comme étant des détails. Nous avons ainsi un algorithme récursif (programmé itérativement pour plus d'efficacité) permettant l'affichage d'une cellule et de ses fils. Ce principe est le même pour les autres affichages. L'algorithme s'adapte très facilement aux groupes, puisque ceux-ci sont aussi définis comme des orbites. Nous disposons donc un affichage permettant à la fois une vue hiérarchique des détails et une vue des multipartitions.

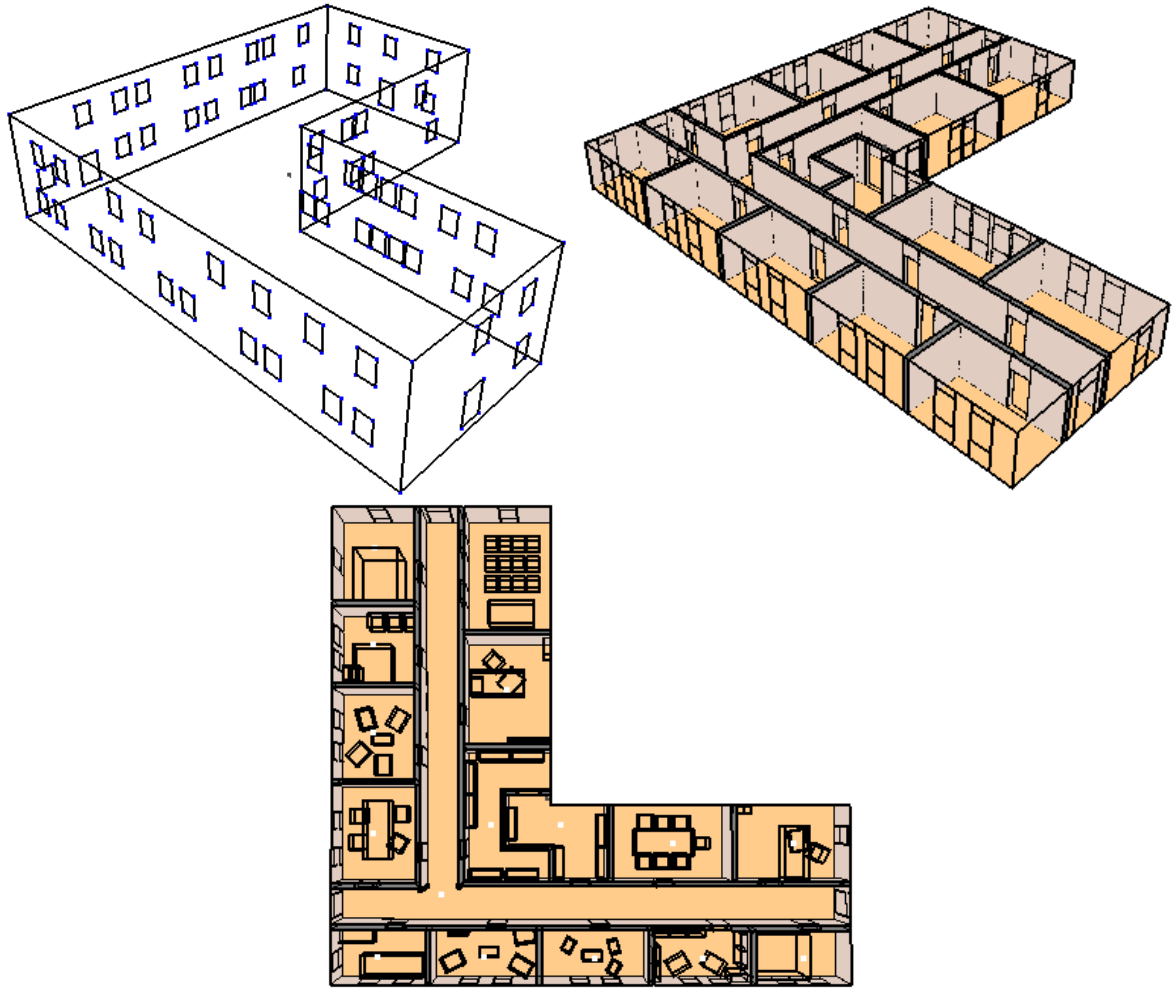
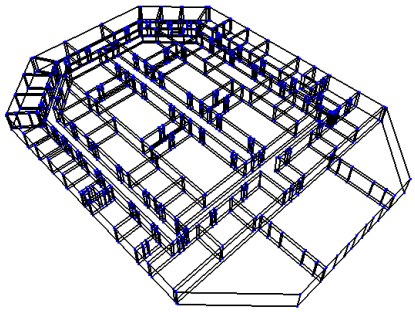
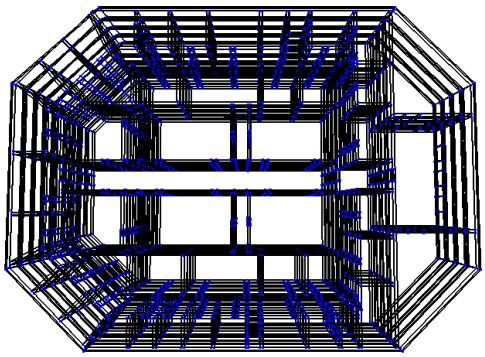


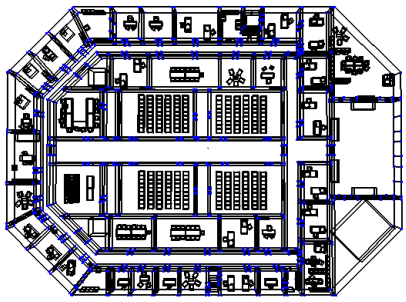
Figure 4.13 – *Bâtiment en forme de L :* 2 *étages,* 27 *pièces et* 175 *meubles soit* 306207 *polygones.*



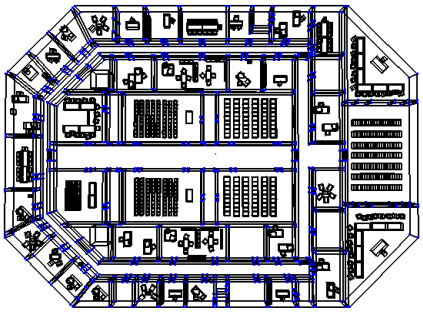
Murs du rez-de-chaussée modélisé manuellement.



b. Copie de cet étage dans les trois autres étages du bâtiment octogonal.



c. Ameublement manuel du rez-de-chaussée.



d. Copie des pièces et utilisation des scripts pour meubler les autres étages (nombreuses pièces identiques à l'étage *(c)*).

Figure 4.14 – *Bâtiment octogonal :* 4 *étages,* 232 *pièces et* 2654 *meubles soit* 5 339 422 *polygones.*

Lors de la modélisation, l'affichage de la hiérarchie est souvent utilisé pour visualiser l'aménagement d'une pièce ou la modifier. Il est rare de l'utiliser pour voir le bâtiment au complet, car dans ce cas, le nombre de primitives graphiques à afficher est tel (plusieurs millions de polygones) que les parcours ne sont plus suffisamment rapides (plusieurs secondes) et que la carte graphique ne permet plus un taux de rafraîchissement correct. Pour la modélisation, nous avons donc choisi de ne jamais afficher les meubles, mais seulement leurs boîtes englobantes, à l'exception d'une demande explicite de l'utilisateur. Le résultat d'un affichage de la structure du bâtiment (excepté les meubles) peut se voir en fil de fer sur la figure 4.14 et en polygonal sur les figures 4.11 et 4.9. Néanmoins, pour la visualisation réaliste d'environnements complexes, ces

solutions ne sont pas suffisantes. Nous avons développé une série d'algorithmes de visualisation et de simulation d'éclairage permettant de prendre une telle masse de données (chapitres 5 et 6).

CHAPITRE 5

VISUALISATION PAR LANCER DE RAYONS

Une fois les bâtiments modélisés avec un maximum d'informations géométriques, sémantiques et topologiques, nous souhaitons générer des images réalistes de ces complexes architecturaux. Nous devons donc simuler le transport de l'énergie lumineuse à travers des scènes possédant plusieurs millions de polygones.

À l'heure actuelle, les processeurs de carte graphique tels que le NV40 de nvidia peuvent traiter d'après les documentations techniques jusqu'à 600 millions de triangles par seconde soit une scène de 25 millions de triangles à un taux de rafraîchissement de 24 images par secondes sans illumination. Mais peu de cartes en sont encore munies et plus les processeurs peuvent gérer de faces, plus les modèles se compliquent. De nombreux travaux ont proposé des méthodes de manipulation de grandes masses de données pour le tampon de profondeur (Z-Buffer) [TS91, FST92, MS95, EMB01]. D'autres techniques de visualisation comme le lancer de rayons ont aussi été revisitées, Wald et al. [WPS$^+$03] proposent la bibliothèque OpenRT permettant de tracer 4 millions de rayons par seconde dans une scène de $1,5$ millions de triangles soit un taux de rafraîchissement de 13 images par seconde pour une image $640x480$ sur un Pentium IV $2,5$ GHz. Il a aussi développé cet algorithme sur des clusters de PC. Il s'agit des meilleurs résultats obtenus actuellement en lancer de rayons. Des études en simulation d'éclairage ont été réalisées pour des complexes architecturaux ; les temps de calcul sont considérablement plus longs, mais il s'agit d'une simulation très fine des trajets lumineux [TFFH94, MBSB03].

L'avantage de notre modèle topologique comparé aux structures accélératrices en visualisation réside dans la richesse des informations qu'il contient. La représentation par partitions de la scène peut être utilisée pour sélectionner un ensemble d'éléments à visualiser, par exemple lorsqu'un utilisateur ne souhaite afficher qu'un étage du bâtiment. Cette fonctionnalité est d'ailleurs très largement utilisée au cours de la modélisation. De plus, la représentation hiérarchique du bâtiment contient toutes les informations nécessaires à la simplification des calculs de visibilité.

La première technique que nous avons abordée en visualisation a été le tampon de profondeur (en anglais *Z-Buffer*) afin de pouvoir afficher le bâtiment édité, comme nous l'avons vu dans la partie modélisation. Ce travail a été réalisé parallèlement au développement du modeleur. L'affichage Z-Buffer est intéressant lors du processus de modélisation, mais apporte peu en terme de réalisme. Nous nous sommes donc intéressés au calcul d'images synthétiques de grands bâtiments. Le tracé de rayons [App68] est souvent utilisé pour traiter les problèmes de visibilité, y

compris pour les méthodes de simulation d'éclairage. Il consiste à créer des rayons (demi-droite) dans une scène et à calculer les intersections avec les objets. C'est cette technique que nous avons choisi de développer en priorité sur notre structure topologique dans le but ensuite de créer un lancer de rayons [Whi80] (pour la prise en compte des sources lumineuses). Le tracé de rayon est aussi utilisé pour le lancer de photons [JC95] pour les calculs de trajectoire des photons et la génération de l'image finale, ou en calculs de radiosité [GTGB84] pour les calculs de visibilité entre faces et le *final gathering*.

Autour de notre structure de données à base de cartes généralisées, nous avons développé plusieurs algorithmes de tracé de rayons dans un environnement architectural complexe. Savoir rapidement calculer les intersections d'un rayon dans un grand bâtiment nous a permis ensuite d'adapter les techniques trouvées à un algorithme d'éclairement global : le lancer de photons. Ce chapitre expose les démarches et résultats obtenus en lancer de rayons et le chapitre suivant s'intéresse au lancer de photons.

Dans un premier temps, nous avons développé un algorithme de tracé de rayons exploitant directement la structure topologique à l'aide de parcours d'orbites et de parcours de la hiérarchie. Néanmoins, les temps de calcul sont très longs. En effet, le noyau topologique sur lequel repose notre structure est dédié à la modélisation. Les opérations de parcours sont fiables et suffisamment rapides pour une modélisation interactive. Néanmoins, pour la visualisation, il est difficile d'accéder instantanément aux notions de faces et de volumes. Les cartes généralisées n'offrent ces informations que sous forme d'orbites et de plongements et demandent de nombreux parcours pour réunir les données. Aucun accès direct à tous les plongements des sommets (coordonnées) et des faces (couleurs, textures) n'est possible sans un parcours de brins. La recherche des plongements lors de chaque calcul d'intersection ralentit donc d'une manière critique les temps de calcul.

Nous avons donc fait d'autres choix d'implantation des algorithmes de visualisation et simulation d'éclairage en attendant de pouvoir développer un noyau topologique dédié à la visualisation. Les deux autres algorithmes développés sont les suivants :

- nous avons réalisé un prétraitement pour accélérer le temps de calcul d'une image avec le logiciel POV-Ray [1]. La relative lenteur de notre tracé de rayons fonctionnant directement sur la structure est compensée en ne faisant des tests d'intersection qu'avec les faces des murs pour créer une liste de pièces visibles par l'observateur.
- au vu des temps de calcul actuels (lancer de rayons interactif sur certaines scènes [WPS+03]), nous avons ensuite préféré développer notre propre lancer de rayons pour réduire le temps de calcul. La structure topologique a alors servi de support pour générer une structure de graphe d'adjacence des pièces.

Avant d'aborder les différentes techniques de rendu auxquelles nous nous sommes intéressés (le lancer de rayons et le lancer de photons), nous rappelons les grandeurs, dites *photométriques*, intervenant dans les modèles physiques d'éclairement global. Nous exposons ensuite l'équation de luminance permettant de décrire les échanges lumineux entre les surfaces d'une scène. Dans un cadre réaliste, cette équation intégrale ne possède pas de solution analytique. Elle peut néan-

[1] *The terms "POV- Ray" and "Persistence of Vision Raytracer" are trademarks of the Persistence of Vision Development Team. http ://www.povray.org*

moins être discrétisée pour obtenir différentes solutions approchées. Nous montrons une simplification de cette équation donnant lieu à l'équation de lancer de rayons (sous-section 5.2.1) et une autre dans le chapitre suivant pour le lancer de photons (sous-section 6.1.1). Nous présentons ensuite le fonctionnement d'un lancer de rayons. Enfin, nous proposons les deux techniques optimisées de tracé de rayons et le lancer de rayons que nous avons implantées, ainsi que leurs résultats respectifs.

5.1 Les grandeurs physiques et l'équation de luminance

La lumière peut être considérée comme un phénomène ondulatoire et particulaire. Par exemple, la diffraction lumineuse relève de l'aspect ondulatoire de la lumière. Les particules lumineuses, baptisées *photons* par Einstein, ont été mises en évidence grâce aux phénomènes photo-électriques (par exemple, un métal se charge électriquement lorsqu'il est exposé au soleil). Cette dualité est également présente au niveau des algorithmes de simulation d'éclairage. Les traitements sont réalisés sur un ensemble de longueurs d'onde pour le spectre visible et souvent les relations de visibilité sont déterminées à l'aide d'un tracé de rayons correspondant au trajet de certaines particules.

5.1.1 Le spectre électromagnétique

La radiométrie est la science de la mesure physique de l'énergie électromagnétique. La photométrie consiste à mesurer la sensation visuelle produite par les ondes électromagnétiques. En effet, l'œil humain n'est sensible qu'aux longueurs d'onde comprises entre l'ultraviolet ($\approx 380nm$) et l'infrarouge ($\approx 770nm$). La figure 5.1 met en évidence la partie visible du spectre électromagnétique et fait correspondre les couleurs avec les différentes longueurs. Dans la suite, les longueurs d'onde sont notées λ. Les calculs d'illumination doivent en principe s'effectuer pour chaque longueur d'onde du domaine visible.

En général, l'espace couleur RVB (Rouge Vert Bleu) est utilisé, car il permet de réaliser les calculs directement pour des valeurs affichables par les moniteurs d'ordinateur. Mais certaines approches de simulation d'éclairage ont pour souci de prendre en compte les phénomènes ondulatoires le plus rigoureusement possible. Pour cela, les échanges lumineux entre surfaces sont décrits à l'aide d'un échantillonnage spectral beaucoup plus précis. Par exemple, Meneveaux [Men98] utilise un échantillon spectral de 80 valeurs. Le passage vers l'espace de couleur RVB est réalisé a posteriori juste avant l'affichage de l'image.

Pour nos algorithmes comme pour la plupart des algorithmes de visualisation, nous avons utilisé l'espace RVB même si nous restons conscients que cela mène à des résultats inexacts. L'objet de notre travail ne se situe pas à ce niveau de réalisme et nos algorithmes restent compatibles avec la prise en compte d'un plus grand nombre de longueurs d'onde. Par ailleurs, utiliser seulement 3 valeurs permet de réduire l'espace mémoire nécessaire au traitement.

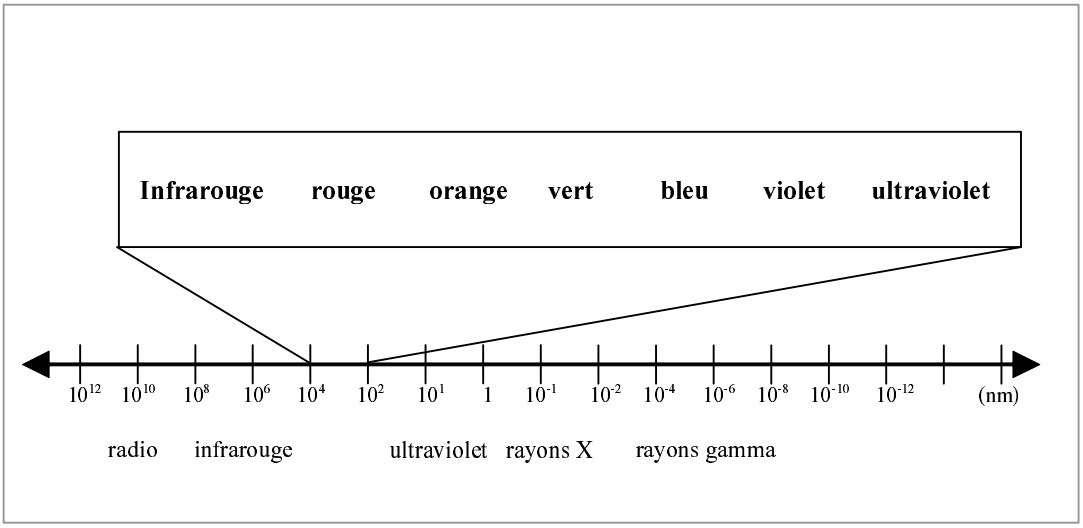


Figure 5.1 – *L'œil perçoit les longueurs d'ondes comprises entre l'infrarouge et l'ultraviolet, mises en évidence ici dans le cadre*

5.1.2 L'angle solide

Un *angle solide* est en trois dimensions ce qu'est un angle en deux dimensions. Il est utilisé pour mesurer la portion d'espace occupé par un objet S vu depuis une position donnée x (figure 5.2). Noté $d\vec{\omega}$, il délimite un cône dans l'espace. Il se mesure en stéradians (sr) et se calcule en déterminant l'aire de la projection centrale de la surface S sur la sphère unité centrée en x. Dans la suite, les vecteurs directeurs sont notés $\vec{\omega}$ pour conserver une cohérence de notation avec la direction de l'angle solide.

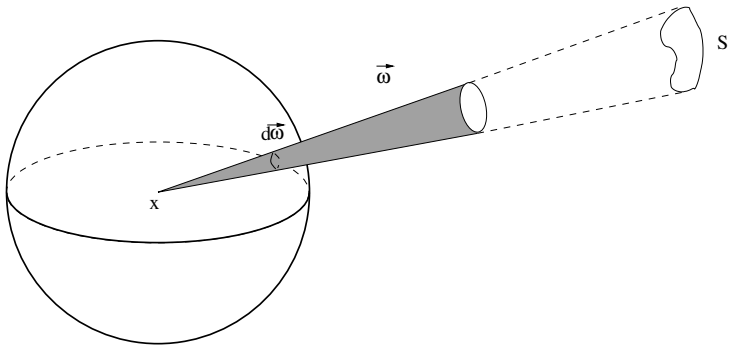


Figure 5.2 – *L'angle solide $d\vec{\omega}$ correspond à l'aire de la projection de la petite surface S sur la sphère unité centrée en x.*

La sphère est décrite à l'aide des coordonnées sphériques (θ et ϕ sur la figure 5.3). L'angle θ est l'angle polaire (angle formé avec la normale) et ϕ est l'angle azimutal (angle formé avec l'axe des x du repère local de la face). L'angle solide élémentaire peut s'écrire ainsi :

$$d\vec{\omega} = sin\theta d\theta d\phi \tag{5.1}$$

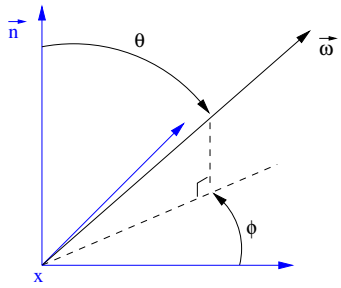


Figure 5.3 – *La direction $\vec{\omega}$ peut s'exprimer en fonction des coordonnées sphériques θ et ϕ.*

5.1.3 Flux, intensité et luminance

Les objets d'une scène peuvent émettre un rayonnement lumineux ou en recevoir. Afin de quantifier ces échanges lumineux, un ensemble de définitions a été proposé en radiométrie. Le *flux*, noté Φ, permet d'exprimer la quantité d'énergie émise ou reçue par unité de temps par un objet. Il s'exprime en Watts (W). L'*intensité*, notée I, permet de caractériser l'énergie émise par unité d'angle solide en fonction d'une direction donnée. Elle utilise donc la notion d'angle solide. L'intensité représente le flux émis par unité d'angle solide et s'exprime en Watts par stéradian (W/sr).

$$I = \frac{d\Phi}{cos\theta d\vec{\omega}} \tag{5.2}$$

L'intensité décrit la quantité d'énergie transitant dans une direction donnée. Pour décrire ce que perçoit réellement l'œil humain lorsqu'une source éclaire un objet, il faut s'intéresser à la *luminance* (en anglais *radiance*) qui décrit le flux (reçu ou émis) par unité d'angle solide et par unité de surface.

$$L = \frac{d^2\Phi}{cos\theta d\vec{\omega} dA} \tag{5.3}$$

5.1.4 La BRDF

Les grandeurs permettant de décrire les transferts lumineux ont été décrites, les faces des objets doivent maintenant être caractérisées afin de savoir de quelle manière l'énergie reçue est renvoyée. Pour cela, un matériau est caractérisé par une fonction de distribution de la réflectance bidirectionnelle, en anglais : BRDF, *Bidirectional Reflectance Distribution Function*. Pour une

direction d'incidence ou d'éclairement $\vec{\omega}_i$ et une direction de réflexion ou d'observation $\vec{\omega}_0$, cette fonction exprime le rapport entre la luminance réfléchie d'un petit élément de surface dA autour d'un point x et l'éclairement incident E_i à celle-ci (et ceci, pour chaque longueur d'onde donnée). Dans les équations suivantes, la BRDF est notée f_r.

$$f_r(x, \vec{\omega}_0, \vec{\omega}_i) = \frac{dL_r(x, \vec{\omega}_0)}{dE_i(x, \vec{\omega}_i)} \tag{5.4}$$

Elle a été introduite par Nicodemus *et al* [NRH+77] et suppose que la lumière arrivant en un point d'une surface est réfléchie en ce même point. Le résultat de cette fonction est le pourcentage de luminance réémise par rapport à l'éclairement reçu pour chaque couple de directions $((\theta_i, \phi_i), (\theta_r, \phi_r))$. Pour calculer la valeur de la luminance élémentaire dL dans une direction donnée en fonction d'une luminance incidente L_i, il suffit alors d'exprimer l'éclairement incident en fonction de la luminance L_i pour obtenir :

$$dL(x, \vec{\omega}_0) = f_r(x, \vec{\omega}_0, \vec{\omega}_i) L_i(x, \vec{\omega}_i)(\vec{\omega}_i \cdot \vec{n}) d\vec{\omega}_i \tag{5.5}$$

Avec $\vec{n}$ la normale à la surface au point x. Une propriété physique importante de la BRDF est la loi de conservation de l'énergie. Une surface ne peut pas réfléchir plus de lumière qu'elle n'en reçoit et la BRDF doit satisfaire l'équation suivante :

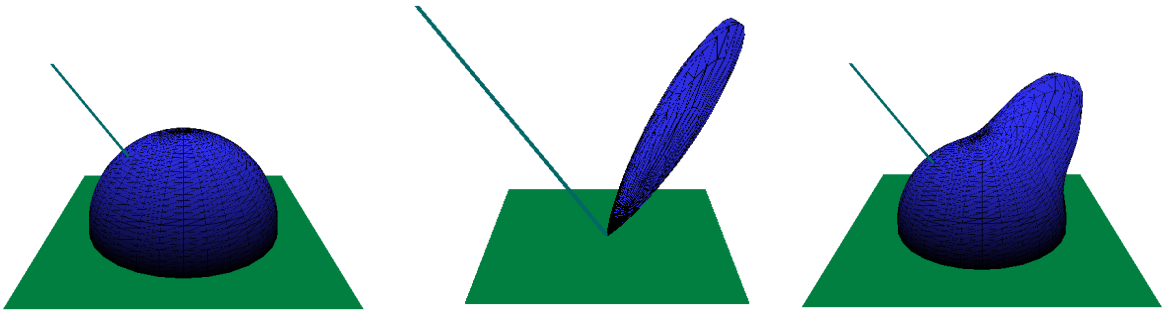
$$\int_{\Omega} f_r(x, \vec{\omega}_0, \vec{\omega}_i)(\vec{\omega}_i \cdot \vec{n}) d\vec{\omega}_i \leq 1, \forall \vec{\omega}. \tag{5.6}$$

Les valeurs de la BRDF d'une surface sont très nombreuses : à chaque couple (point, direction d'incidence) correspond un ensemble de valeurs, une par direction d'émission. Sur la figure 5.4, des représentations 3D montrent dans chaque direction la valeur de la luminance obtenue pour une direction d'incidence donnée et un point fixé sur une surface. Ceci nous donne une bonne idée de la complexité de la BRDF et du coût de stockage.

Dans la littérature de nombreux modèles de BRDF ont été proposés [Pho75, CT82, War92, Lew94]. Nous ne décrivons ici que deux modèles simples et très connus : Lambert et Phong. Le modèle de Lambert est le modèle le plus simple, car il considère la BRDF comme constante (figure 5.4.a). La luminance émise par une surface est la même dans toutes les directions. Les surfaces ayant cette BRDF sont dites *lambertiennes* ou *parfaitement diffuses*. Ce type de BRDF respecte la loi de conservation de l'énergie et la loi de réciprocité de Helmholtz. Il est en général utilisé dans les calculs de radiosité, parce qu'il permet de simplifier les équations caractérisant les échanges lumineux. C'est aussi ce type de BRDF que nous utilisons actuellement au sein de nos algorithmes, même si ces derniers sont prévus pour prendre en compte des BRDF plus complexes.

$$f_r(x, \vec{\omega}_0, \vec{\omega}_i) = K_d \tag{5.7}$$

Dans la réalité, aucune surface n'est lambertienne. Les surfaces ont tendance à réfléchir la lumière : ce phénomène est appelé spécularité. Toutes les surfaces sont plus ou moins spéculaires et un miroir est une surface spéculaire parfaite. Phong propose un modèle de BRDF permettant



a. b. c.

Figure 5.4 – *Représentations graphiques de la luminance émise par une surface ayant reçue de l'énergie depuis une direction fixée. Ces graphiques utilisent le modèle de Phong : a. une surface lambertienne. - b. une surface purement spéculaire. - c. une surface classique, mêlant diffus et spéculaire.*

de prendre en compte les surfaces diffuses et spéculaires à l'aide de seulement trois coefficients. K_d est le coefficient diffus de la surface et K_s son coefficient spéculaire. Les valeurs de ces deux coefficients sont comprises entre 0 et 1.

$$f_r(x, \vec{\omega}_0, \vec{\omega}_i) = K_d + K_s \frac{cos^n \phi}{cos\theta} \tag{5.8}$$

Ici $cos\theta$ représente le produit scalaire $(\vec{\omega}_i \cdot \vec{n})$. Une surface parfaitement diffuse a pour coefficients : $K_d = 1$ et $K_s = 0$ (figure 5.4.a). Un miroir a pour coefficients : $K_d = 0$ et $K_s = 1$ (figure 5.4.b). Le coefficient de spécularité n, ou indice de brillance, permet d'ajuster la taille du lobe spéculaire. Le rôle de chaque coefficient est visible sur la figure 5.5. Ce modèle n'est pas correct puisqu'il ne respecte pas la loi de conservation de l'énergie ; une face peut renvoyer plus de lumière qu'elle n'en reçoit si les coefficients sont mal choisis. Lewis [Lew94] a proposé en 1994 un modèle de Phong modifié, physiquement correct.

$$f_r(x, \vec{\omega}_0, \vec{\omega}_i) = \frac{K_d}{\pi} + \frac{(n+2)K_s}{2\pi} \cos^n \phi$$

5.1.5 L'équation de luminance

Une fois les grandeurs physiques définies et les surfaces caractérisées, nous pouvons aborder l'équation de luminance qui exprime les échanges lumineux dans un environnement non participant. Ceci signifie que le milieu dans lequel les objets sont placés est totalement transparent (pas de fumée, de poussière, de liquide, etc.). La première adaptation des modèles originaires de la physique à la synthèse d'images a été proposée par Kajiya en 1986 [Kaj86]. Il pose une équation, dite équation de rendu, exprimant les échanges lumineux entre les surfaces de la scène. Kajiya prouve aussi que les méthodes de radiosité et de Monte Carlo sont deux estimations de cette même équation. L'équation de rendu peut aussi s'exprimer en terme de luminance [Laf96] :

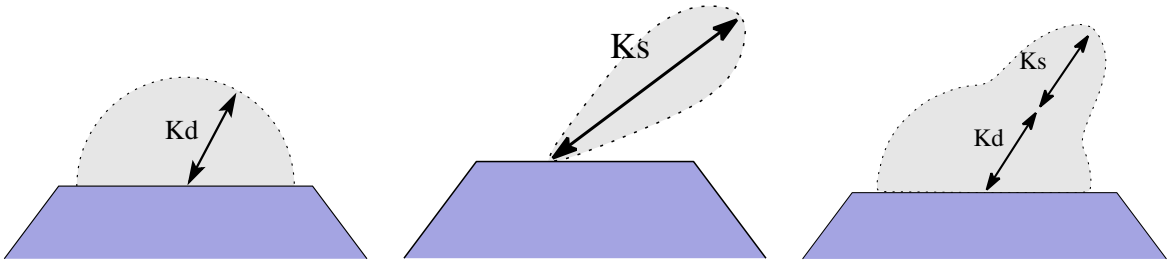


Figure 5.5 – *Mise en évidence des coefficents K_d et K_s des trois types de BRDF illustrées sur la figure précédente. Le coefficient n n'est pas représenté ici : a. Une surface lambertienne avec $K_s = 0$ et $K_d = 1$. - b. Une surface uniquement spéculaire avec $K_s = 1$, $K_d = 0$ et la taille de l'arrondi du lobe spéculaire dépend de l'exposant de spécularité n. - c. Une surface à la fois diffuse et spéculaire.*

équation 5.9. L'équation de luminance exprime alors la luminance émise par une surface en un point x suivant une direction $\vec{\omega}_0$, soit la quantité de lumière d'un point x perçue par un œil placé dans la direction $\vec{\omega}_0$.

$$\begin{aligned} L(x,\vec{\omega}_0) &= L_e(x,\vec{\omega}_0) + L_r(x,\vec{\omega}_0) \\ &= L_e(x,\vec{\omega}_0) + \int_{\Omega} L_i(x,\vec{\omega}_i) f_r(x,\vec{\omega}_0,\vec{\omega}_i)(\vec{\omega}_i \cdot \vec{n}) d\vec{\omega}_i \end{aligned} \tag{5.9}$$

La luminance (schématisée sur la figure 5.6) est la somme de la luminance propre L_e émise par la face dans cette direction (non nulle pour une source lumineuse) et de la luminance L_r réfléchie au point x (de normale $\vec{n}$) dans la direction $\vec{\omega}_0$. La luminance réfléchie s'exprime elle-même comme l'ensemble des énergies lumineuses arrivant au point x depuis une direction $\vec{\omega}_i$ et repartant dans la direction $\vec{\omega}_0$. Il s'agit donc d'une intégrale sur tout l'hémisphère centré en x noté Ω.

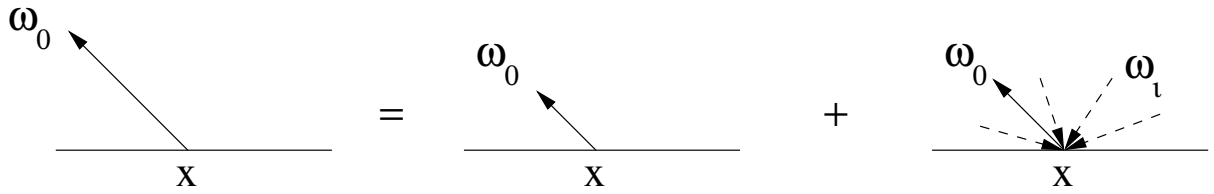


Figure 5.6 – *La luminance émise par une surface dans une direction $\vec{\omega}_0$ est la somme de sa luminance propre (si il s'agit d'une source lumineuse) et de l'ensemble des énergies lumineuses reçues au point x et réfléchies dans la direction $\vec{\omega}_0$.*

5.2 Principe du lancer de rayons

Afin de calculer l'ensemble de ces échanges lumineux, les algorithmes utilisent souvent des techniques de tracé de rayons (*ray casting*). Les rayons permettent de définir les relations de visibilité entre l'œil et les faces de la scène, entre une face et d'autres faces (mais pas de manière exacte) ou encore entre une face et des sources lumineuses. Si le rayon atteint une face entre le point de départ et le point d'arrivée, ils sont masqués l'un pour l'autre. Le tracé de rayons est utilisé pour le lancer de rayons récursif (*ray tracing*) et dans des algorithmes dits d'éclairement global tels que les méthodes de Monte Carlo ou les calculs de radiosité.

Le lancer de rayons permet d'estimer l'éclairage d'un objet en essayant de retrouver le cheminement de la lumière depuis la source en partant l'œil. Pour cela, des rayons sont tracés depuis l'œil vers le plan de l'image situé dans la scène ; ces rayons sont appelés *rayons primaires* (figure 5.7). Chaque rayon est examiné pour déterminer une intersection avec un des objets de la scène. Si des intersections sont trouvées, une recherche de la plus proche permet de déterminer la face visible et ainsi de calculer la couleur du pixel à l'aide d'une simplification de l'équation de luminance. Si le rayon manque tous les objets, le pixel est mis à la couleur de fond.

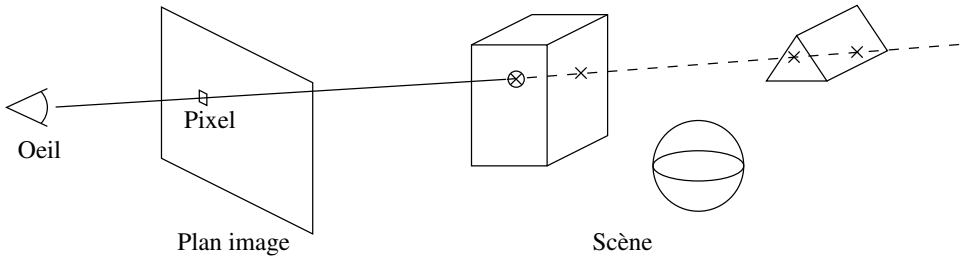


Figure 5.7 – *Le rayon primaire permet de trouver pour un pixel donné l'intersection la plus proche.*

Les ombres et les réflexions spéculaires sont prises en compte en lançant de nouveaux rayons vers les sources et dans la direction spéculaire. Ces rayons sont appelés *rayons secondaires*. Lorsqu'un rayon rencontre une face dont le coefficient spéculaire est non nul, ce type de rayon permet de déterminer quelle face est vue dans la direction miroir (direction symétrique par rapport à la normale) afin de créer un effet de reflet. Ces rayons sont lancés récursivement jusqu'à un niveau de profondeur fixé. Cet algorithme appelé *lancer de rayons récursif* a été proposé par Whitted en 1980 [Whi80]. Les rayons partant de la face vers les sources lumineuses sont appelés *rayons d'ombre*.

Le trajet de la lumière étant retracé en partant de l'œil jusqu'aux sources lumineuses, ceci rend difficile une simulation poussée des échanges lumineux tels que les inter-réflexions. Ces dernières peuvent être prises en compte par d'autres techniques : le tracé de chemins, le calcul de radiosité ou encore le lancer de photons décrit dans le chapitre suivant.

5.2.1 L'équation de luminance adaptée au lancer de rayons

Pour chaque pixel de l'image, un rayon est lancé depuis l'oeil (rayon primaire). Une fois, l'intersection la plus proche de l'œil trouvée, nous connaissons le point d'intersection x pour lequel nous devons estimer la luminance perçue par l'observateur. Rappelons cette équation :

$$L(x, \vec{\omega}_0) = L_e(x, \vec{\omega}_0) + \int_{\Omega} L_i(x, \vec{\omega}_i) f_r(x, \vec{\omega}_0, \vec{\omega}_i)(\vec{\omega}_i \cdot \vec{n}) d\vec{\omega}_i \tag{5.10}$$

Pour un lancer de rayons, les sources lumineuses sont fréquemment considérées comme ponctuelles afin de ne tracer qu'un rayon lumineux par source. Les faces de la scène sont considérées non lumineuses. D'où $L_e(x, \vec{\omega}_0) = 0$ pour tout point x sur une face. Pour calculer le résultat de l'intégrale, nous séparons les différents chemins en deux catégories comme le montre la figure 5.8 : les trajets directs et les trajets indirects. Un trajet direct est un trajet lumineux allant directement de la source au point x. Les trajets indirects sont les trajets passant par d'autres faces et sont eux-mêmes divisés en deux catégories : les réflexions et les inter-réflexions. Les réflexions de la lumière dues aux faces spéculaires sont calculées de manière récursive : un rayon dans la direction miroir est lancé dans la scène et un nouveau point d'intersection x' est trouvé pour lequel l'illumination est calculée. Nous rappelons que le cadre de l'étude se limite aux inter-réflexions diffuses. Nous ne traitons pas la composante spéculaire des matériaux, toutes les faces sont supposées lambertiennes. L'intégrale est donc divisée donc en deux parties : illumination directe et illumination indirecte.

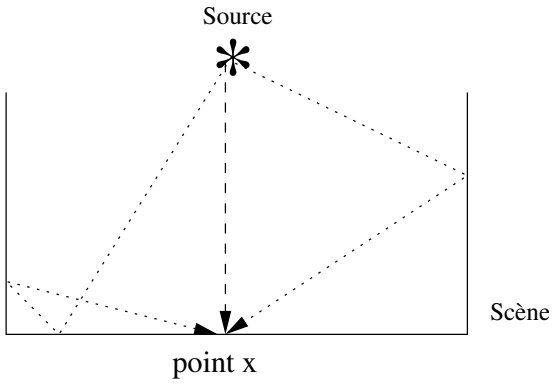


Figure 5.8 – *Illustration de différents trajets pris par la lumière pour atteindre un point. Plus le trajet est compliqué, moins l'énergie lumineuse est importante. Au centre en tireté, le trajet est direct depuis la source vers le point x. De chaque côté en pointillés, un exemple de trajet indirect est proposé.*

Dans le chapitre suivant, nous expliquons une méthode permettant de prendre en compte les inter-réflexions diffuses. Le calcul des inter-réflexions étant très coûteux, il est habituel en lancer de rayons de les estimer grossièrement à l'aide d'une constante appelée *terme ambiant*. Une valeur empirique $L_{ambiant}$ est souvent utilisée pour remplacer la luminance totale des inter-réflexions, les directions d'incidence n'étant pas prises en compte. Cette valeur est multipliée

par K_d, le coefficient diffus, pour exprimer la quantité d'énergie correspondant au terme ambiant que la surface peut réfléchir. Pour le calcul d'illumination directe, seules les sources ponctuelles directement visibles depuis le point x sont prises en compte. Posons $L_{d,i}$ la luminance incidente en un point x depuis une source lumineuse ponctuelle, l'équation de luminance peut donc se récrire ainsi :

$$L(x, \vec{\omega}_0) = K_d L_{ambiant} + \int_{\Omega} L_{d,i}(x, \vec{\omega}_i) K_d (\vec{\omega}_i \cdot \vec{n}) d\vec{\omega}_i \tag{5.11}$$

Remarquons que le flux émis par une source lumineuse de puissance Φ_s dans une direction $d\omega$ s'écrit :

$$d\Phi = \frac{\Phi_s}{4\pi} d\omega \tag{5.12}$$

Avec $d\omega$ l'angle solide correspondant à l'élément de surface vue depuis la source. Ceci se retrouve en remarquant que la luminance partant de la source suivant un angle solide donné $d\omega$ correspond à un pourcentage du flux total et que ce pourcentage se détermine en divisant l'ensemble des $d\omega$ (i.e. l'aire de la sphère, 4π) par l'angle solide concerné.

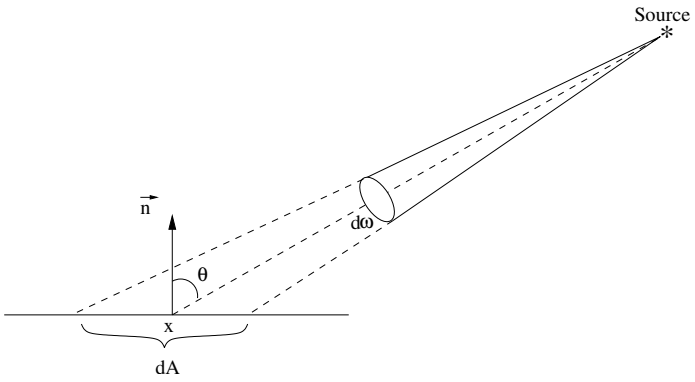


Figure 5.9 – *L'angle solide $d\omega$ correspondant à la surface dA.*

L'angle $d\omega$ peut aussi s'exprimer en fonction de l'élément de surface dA, comme le montre la figure 5.9, de la manière suivante :

$$d\omega = \frac{dA cos\theta}{r^2} = \frac{dA(\vec{\omega}_i \cdot \vec{n})}{r^2} \tag{5.13}$$

Avec r la distance entre x et la source. Le flux émis par la source vers un élément de surface dA est donc :

$$d\Phi = \frac{\Phi_s}{4\pi r^2} (\vec{\omega}_i \cdot \vec{n}) dA \tag{5.14}$$

Connaissant ce flux et avec l'équation 5.3, l'intégrale peut donc se simplifier en une somme finie des contributions de chaque source lumineuse de la manière suivante :

$$L(x, \vec{\omega}_0) = K_d L_{ambiant} + \sum_{sources\ visibles} K_d \frac{\Phi_s}{4\pi r^2} (\vec{\omega}_i \cdot \vec{n}) \tag{5.15}$$

Dans la pratique, afin de savoir si une source lumineuse est visible depuis le point x, un rayon d'ombre est envoyé depuis le point d'intersection vers chaque source lumineuse de la scène. Si ce rayon rencontre un objet avant d'atteindre la source, cette dernière n'est pas prise en compte pour le calcul d'illumination (figure 5.10). Enfin, pour prendre en compte les réflexions et aussi la transparence, des rayons dits secondaires peuvent être émis pour prendre en compte l'effet miroir (réflexion) sur une surface spéculaire ou la transparence (réfraction) d'un volume.

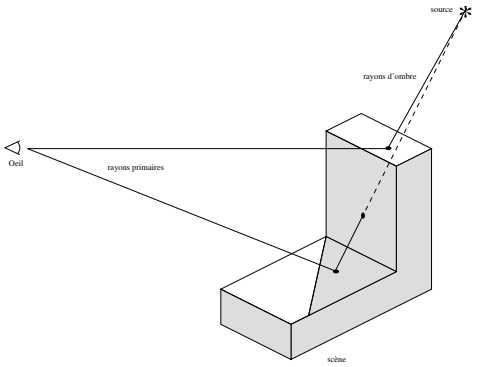
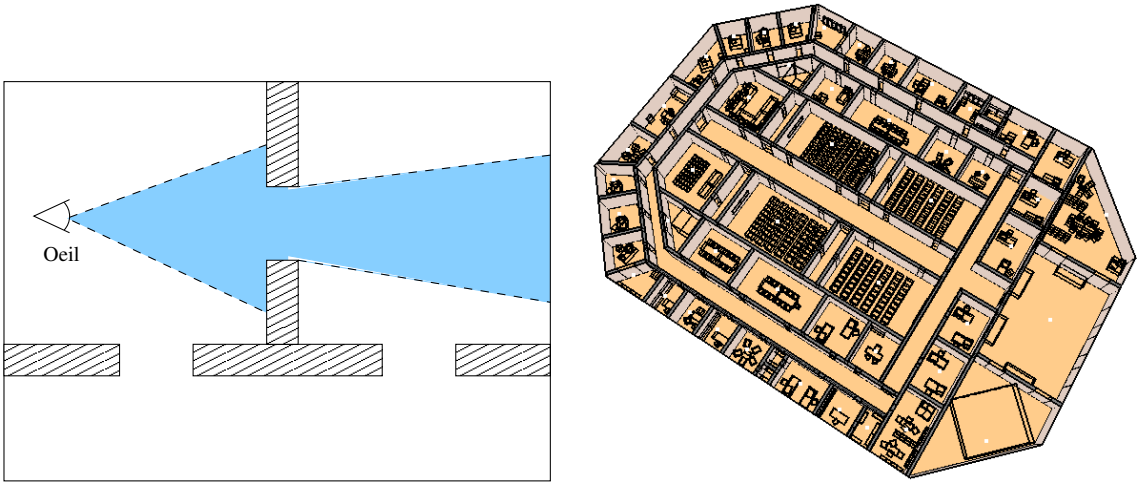


Figure 5.10 – *Un rayon d'ombre est lancé vers la source lumineuse pour déterminer si le point d'intersection est à l'ombre ou non.*

5.3 Le tracé de rayons à l'aide de notre structure topologique

Notre premier travail consiste à programmer un tracé de rayons utilisant directement le modèle hiérarchique sans structure accélératrice. Cet algorithme servira de base pour l'ensemble de nos algorithmes de visibilité et de simulation d'éclairage. L'objectif est d'avoir un algorithme rapide permettant de tracer un rayon dans un bâtiment complexe de plusieurs millions de polygones afin de déterminer l'ensemble des faces visibles depuis un point de vue. La figure 5.11.a montre que toutes les pièces d'un bâtiment ne sont pas toujours visibles depuis un point de vue donné et que parmi les pièces visibles seule une partie de ces pièces entre dans le champ de vision. Il n'est donc pas utile pour ce calcul de traiter la totalité du bâtiment. Nous avons donc cherché à optimiser le tracé d'un rayon dans un bâtiment complexe tel que celui de la figure 5.11.b.

La totalité d'un bâtiment de plusieurs millions de polygones ne loge pas en mémoire. Néanmoins, nous pouvons remarquer sur le tableau 5.12 que l'ensemble des polygones décrivant sa structure (les murs) ne forme qu'un petit pourcentage du nombre total de faces. La description des meubles dans nos bâtiments complexes forme en général plus de 95% du nombre total de



a. b.

Figure 5.11 – *a. Le champ de vision n'englobant pas l'ensemble du bâtiment, il est inutile d'avoir en mémoire la totalité du bâtiment. - b. Premier étage du grand bâtiment octogonal de 5 millions de polygones.*

	Nombre total de triangles	Murs	Meubles
Bâtiment octogonal	5 339 422	45 547 (0,85%)	5 293 875 (99,15%)
Bâtiment Zarbi	1 074 795	2 278 (0,2%)	1 072 517 (99,8%)
Bâtiment en forme de L	313 325	7 118 (2,3%)	306 207 (97,7%)

Figure 5.12 – *Nombre de polygones composant les trois grands bâtiments que nous avons construit.*

triangles. Il est tout à fait possible de conserver en mémoire la totalité de la structure du bâtiment si les meubles ne résident pas en mémoire. Nous utilisons le modèle hiérarchique présenté dans les chapitres précédents afin de travailler sur les étages et les pièces du bâtiment sans prendre en compte toutes les faces des meubles. Nous modifions donc le principe du tracé de rayons (algorithme 5.13) pour prendre en compte les informations sur le découpage du bâtiment en étages et en pièces.

Dans la structure hiérarchique, le bâtiment est subdivisé en pièces reliées entre elles par des ouvertures et chaque pièce possède ses meubles sous forme de boîtes englobantes et d'une référence vers un fichier stocké sur le disque dur (à translation et rotation près). Nous avons donc à disposition une hiérarchie de volumes englobants. Elle a été engendrée automatiquement lors de la modélisation et peut être exploitée directement. L'algorithme a pour but de réduire le nombre d'intersections à calculer et de travailler avec un minimum de polygones en mémoire. Les intersections ne sont tout d'abord calculées qu'avec les pièces et les boîtes englobantes. Si une intersection est trouvée avec une des boîtes, les intersections avec la liste des faces du meuble sont alors calculées à la manière de Goldsmith et Salmon [GS87]. Remarquons que l'extérieur

```
Pour chaque pixel de l'image faire
    Créer un rayon R allant de l'observateur au centre du pixel ;
    Pour chaque face F de la scène faire
        Calculer l'intersection entre la face F et le rayon R ;
    Fin Pour
    Soit F_min la face dont l'intersection est la plus proche de l'observateur ;
    Si (F_min n'existe pas) alors
        // Il n'y a eu aucune intersection, ceci n'arrive qu'en extérieur
        Affecter la couleur de fond au pixel ;
    Sinon
        // L'intersection la plus proche détermine la couleur du pixel
        Affecter la couleur de F_min au pixel ;
    Fin Si
Fin Pour
```

Figure 5.13 – *Le tracé de rayons classique.*

est considéré comme une pièce particulière (sans niveau de détail et sans meuble) pour conserver l'homogénéité des traitements.

Le principe de tracé de rayons est décrit en pseudo-langage dans l'algorithme 5.14. La première étape consiste à localiser le volume contenant le point de départ des rayons dans la scène : la position de l'observateur. Une fois l'observateur localisé, les rayons primaires sont lancés à travers la pièce de départ. Seuls les meubles lui appartenant sont chargés en mémoire durant le calcul. Une fois le calcul dans cette pièce terminé, la mémoire est libérée. Puisque dans le modeleur, une ouverture est considérée comme un trou dans un mur reliant deux pièces et chaque extrémité de l'ouverture est une face transparente. Lorsqu'un rayon atteint une telle face, nous ne traitons pas immédiatement le rayon, mais nous le stockons dans une mémoire tampon. Cette structure, appelée $File$ dans l'algorithme, est décrite dans la suite. Nous ne propageons le rayon stocké dans la pièce suivante que lorsque tous les rayons ont trouvé une intersection dans la pièce courante. Nous répétons cette propagation des rayons pièce après pièce tant que des ouvertures sont intersectées. Le fait de regrouper plusieurs rayons pour un seul chargement de pièce et de ses meubles limite les échanges entre la mémoire et le disque dur.

Les données stockées dans chaque cellule de la file permettent de mémoriser d'où vient et dans quelle direction va le rayon, dans quelle pièce il se trouve. Remarquons que le stockage du rayon nécessite une autre information : l'emplacement de la dernière intersection trouvée avec une ouverture. Ceci permet d'éviter des boucles infinies dans l'algorithme. La structure d'une cellule de la file est décrite dans 5.15.

En effet, supposons qu'une pièce concave entoure une pièce comme sur la figure 5.16, l'observateur se situe dans le couloir au point 0. Un rayon est tracé dans la direction souhaitée et une face d'ouverture est intersectée au point 1. Nous propageons ce rayon dans la pièce et rencontrons une deuxième ouverture au point 2. Le rayon doit alors de nouveau être propagé dans

```
Trouver dans quelle pièce P_0 se trouve l'observateur ;
Pour chaque pixel de l'image faire
    Créer un rayon R allant de l'observateur au centre du pixel ;
    Ajouter ce rayon avec la pièce P_0 à la file des rayons à traiter File ;
Fin Pour
Charger en mémoire tous les meubles de la pièce P_0 ;
P_i ← P_0 ;
Tant que (File non vide) faire
    // Lecture du rayon à traiter
    R ← Déstocker(File) ;
    // Chargement de la pièce correspondante
    Si (P_i est différente de P_R, la pièce correspondant à R) alors
        Supprimer de la mémoire les meubles de la pièce P_i ;
        P_i ← P_R ;
        Charger en mémoire tous les meubles de la pièce P_i ;
    Fin Si
    Pour chaque face F de la pièce P_i faire
        Calculer l'intersection entre la face F et le rayon R ;
        // Nous ne testons les meubles que si leurs boîtes englobantes sont intersectées
        Si (R intersecte F  Et F fait partie d'une boîte englobante) alors
            // Nous descendons dans la hiérarchie pour tester le meuble
            Appliquer à R les translations et rotations nécessaires ;
            Calculer les intersections entre les triangles du meuble et le rayon R ;
            Appliquer au plus proche point d'intersection les translations et rotations inverses ;
        Fin Si
    Fin Pour
    Soit F_min le triangle dont l'intersection est la plus proche de l'observateur ;
    Si (F_min n'existe pas) alors
        // Seulement possible en extérieur
        Affecter la couleur de fond au pixel ;
    Sinon
        Si (F_min n'appartient pas à une ouverture) alors
            Affecter la couleur de F_min au pixel ;
        Sinon
            // Le rayon va être traité plus tard
            Soit P_adjacente la pièce sur laquelle donne l'ouverture ;
            Stocker dans File le rayon R avec la pièce P_adjacente ;
        Fin Si
    Fin Si

Fait
```

Figure 5.14 – *Notre tracé de rayons propagés avec boîtes englobantes sans source lumineuse.*

```
Struct {
    // Dernière intersection trouvée, initialisée au point de départ du rayon
    Intersection : réel[3] ;
    // Vecteur directeur du rayon
    Direction : réel[3] ;
    // Coordonnée du pixel correspondant au rayon, ici l'image est stockée en ligne
    Pixel : entier ;
    // Numéro de la pièce dans laquelle se propage le rayon, initialisé à P0
    Pièce : entier ;
}
```

Figure 5.15 – *Structure de données d'une cellule de la file stockant les rayons entrants.*

le couloir. Si les points d'intersection successifs ne sont pas mémorisés et si le point d'origine du rayon est toujours le point de départ, l'intersection au point 1 est trouvée en premier et nous créons une boucle infinie. Pour l'éviter, nous stockons donc la dernière intersection trouvée au niveau du rayon.

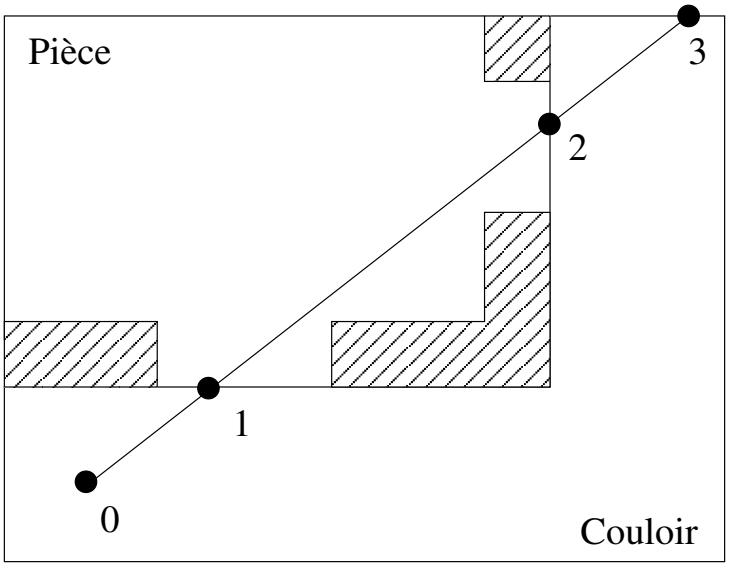


Figure 5.16 – *La propagation d'un rayon nécessite de stocker la dernière intersection trouvée pour éviter une boucle infinie dans l'algorithme.*

Comme nous l'avons vu dans la partie modélisation, le grand nombre de polygones est essentiellement dû aux détails des meubles et non pas à la structure du bâtiment. Avec cet algorithme, nous avons ainsi un faible nombre de meubles chargés en mémoire au même moment. Ce tracé

de rayons nous permet de calculer une image quelque soit le nombre de pièces dans le bâtiment. Des tests ont été effectués sur les bâtiments vus dans le chapitre 3 : le bâtiment en L et l'octogone. Par exemple, le bâtiment octogonal de 5 millions de faces nécessiterait à lui seul près de $2,5$ Go de mémoire vive pour pouvoir résider entièrement en mémoire. Même s'il est techniquement possible d'avoir autant de mémoire vive sur un ordinateur, ceci n'est pas une solution viable à long terme. Nous pouvons toujours générer des bâtiments plus complexes ; une solution matérielle n'est donc pas notre objectif. Notre tracé de rayons a été développé de manière à ce que l'occupation mémoire d'un bâtiment ne dépasse jamais la taille de sa plus grande pièce. Pour l'octogone, la plus grande pièce (450000 triangles) nécessite au total 251 Mo (hiérarchie, données photométriques et interface comprises), notre algorithme ne dépasse donc pas cette taille maximale alors que le bâtiment complet ne loge pas sur 512 Mo de mémoire vive.

Notre tracé de rayons a donc pour avantage de fonctionner quelque soit la taille du bâtiment, ou plus précisément tant que la taille de la plus grande pièce ne dépasse pas celle de l'espace mémoire. En revanche, les temps de calcul de cet algorithme sont très longs. En effet, il faut plus de 30 minutes de calcul pour une image 640 x 480 du bâtiment octogonal sur notre machine de test : AMD 1 GHz, 512 Mo de SDRAM. Nous détaillons ici les différents points de l'algorithme pour identifier les endroits critiques en temps. Nous proposons dans la suite de ce document une solution beaucoup plus rapide. Les trois défauts suivants expliquent les temps de calcul importants de cet algorithme :

- les temps d'accès aux données géométriques dans une carte généralisée ;
- le temps de calcul d'une intersection entre un rayon et un polygone quelconque ;
- la structure de stockage des rayons.

5.3.1 Localisation de l'observateur

La hiérarchie de cartes généralisées découpe l'espace $\mathbb{R}^3$ en volumes. Le principe est donc de localiser dans quel volume l'observateur se trouve : à l'extérieur ou à l'intérieur, à quel étage, dans quelle pièce. Ce calcul se fait très rapidement en plusieurs étapes. La première consiste à tester si le point d'observation est à l'extérieur du bâtiment. Ce test est effectué sur le volume simplifié du bâtiment en racine de la hiérarchie, car il possède toujours un petit nombre de faces. Si le point d'observation est à l'intérieur du bâtiment, il reste à déterminer dans quelle pièce précisément. Pour savoir dans quel étage se trouve l'observateur, seul le profil simplifié des étages est testé : il s'agit du deuxième niveau de la hiérarchie. Si le point est dans un des étages, la troisième étape consiste à chercher dans quelle pièce se trouve l'observateur en utilisant uniquement les pièces décrites par leurs murs au troisième niveau de la hiérarchie.

Afin de savoir si un point se situe dans le volume d'une pièce, d'un étage ou du bâtiment, nous utilisons l'algorithme classique de la demi-droite lancée dans une direction quelconque. Le nombre d'intersections avec le volume détermine la position du point : un nombre pair signifie à l'extérieur, un nombre impair à l'intérieur. La recherche est donc rapide puisque nous ne testons pas tous les volumes du bâtiment, dans le pire cas seul le profil extérieur du bâtiment, les contours des étages et les pièces d'un seul étage sont testés. La localisation est donc une partie peu coûteuse de l'algorithme puisque le nombre d'intersections à calculer est très faible comparé à l'ensemble des autres calculs.

5.3.2 Calcul d'intersection avec des polygones quelconques

Nous avons choisi dans cet algorithme de travailler directement sur la structure hiérarchique, c'est-à-dire de calculer des intersections entre rayons et polygones quelconques issus de la modélisation. Ces polygones peuvent être convexes ou concaves, avec un très grand nombre de sommets (de nombreux couloirs possèdent plus de 50 sommets, le plus grand d'entre eux dans le bâtiment octogonal possède 84 sommets) et des dimensions très différentes (long couloirs, petites ou grandes pièces). Nous devons donc manipuler des données très hétérogènes. Afin de stocker les faces, nous allouons dynamiquement des tableaux de sommets en pré-traitement de l'algorithme pour éviter de devoir rechercher les attributs de chaque cellule au niveau de ses brins. Néanmoins, les calculs d'intersection sont très longs, car les données manipulées ne sont pas contiguës en mémoire et de nombreuses indirections sont nécessaires pour y accéder (passage d'un brin à ses plongements). Il s'agit du point le plus critique de cet algorithme. En effet, l'utilisation d'un logiciel permettant de connaître le temps passé dans chaque fonction (*profiler*) nous montre que 85% du temps de calcul est utilisé pour les tests d'intersections. Ce pourcentage s'explique essentiellement par le grand nombre de faces que contiennent les meubles de la scène.

5.3.3 Le passage d'une pièce à une autre

Les calculs d'intersection rayon/polygone ont lieu au quatrième niveau de la hiérarchie, c'est-à-dire à l'endroit où une pièce est décrite indépendamment des autres avec ses meubles sous forme de boîtes englobantes. Lorsqu'un rayon atteint une face, une information sémantique est vérifiée pour savoir si ce polygone correspond à une ouverture (porte, fenêtre), c'est-à-dire s'il donne sur une autre pièce. Dans ce cas, nous remontons d'un niveau dans la hiérarchie pour connaître l'organisation des pièces entre elles. Le double pointeur père/fils correspondant à la liaison η est utilisé à partir d'un des brins de la face pour passer de la pièce détaillée à l'étage, puis la liaison α_3 permet de passer à travers l'ouverture. Nous avons alors les quelques faces de l'ouverture à tester : actuellement les ouvertures sont très simples, seulement 6 faces. Si le rayon rencontre une face non transparente (un mur), le rayon n'est pas propagé et la couleur de la face est affectée au pixel. Sinon ceci signifie que le rayon est passé dans la pièce suivante, nous récupérons donc cette pièce en passant par α_3 : changement de volume dans la carte généralisée de l'étage. Si cette pièce est détaillée, c'est-à-dire si elle possède un pointeur η défini, nous redescendons dans la hiérarchie. Ce cheminement est illustré sur la figure 5.17. Le rayon est ensuite placé dans la file avec sa nouvelle pièce. Dans le pire cas, nous avons 4 indirections de pointeurs (α_3 et η) et un petit nombre de faces à tester (en général, 4 faces pour une ouverture). Le changement de pièce n'est donc pas un point critique.

5.3.4 Le chargement des meubles

Le chargement d'un meuble demande un accès disque en lecture, plus lent qu'un accès mémoire. L'objet le plus complet que nous ayons est une plante possédant plus de 12000 polygones (exclusivement des triangles pour ce modèle). Le fichier contenant sa description topologique et géométrique fait plus de 2Mo et prend plus d'une demi-seconde à charger. Remarquons que

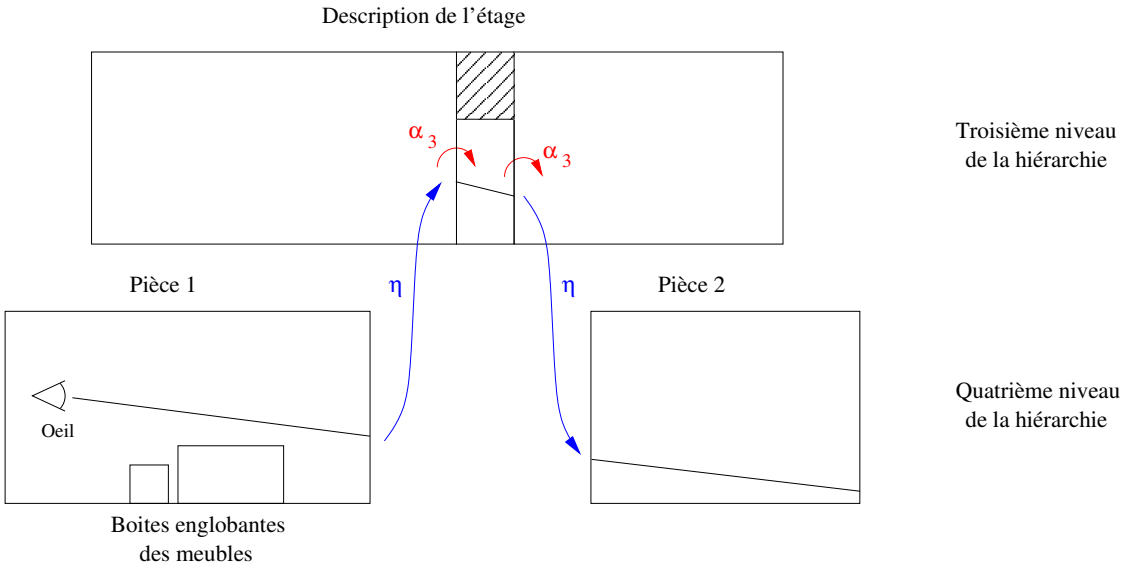


Figure 5.17 – *Lorsqu'une intersection avec une ouverture est trouvée, nous retournons au niveau précédent dans la hiérarchie pour savoir sur quelle pièce donne cette ouverture. L'utilisation des liaisons α_3 et des liaisons hiérarchiques est illustrée ici.*

le chargement de ce même objet sans sa topologie prend moins de 0.4 seconde : simple fichier de listes de faces (fichier NFF). Le chargement des meubles d'une pièce est le second problème identifié dans l'algorithme. Néanmoins, celui-ci est inévitable au vu de la taille que prend un bâtiment complet en mémoire. Il est donc nécessaire de gérer au mieux le changement de pièces pour qu'un meuble soit chargé le moins de fois possible durant tout l'algorithme. La structure de stockage des rayons sortants d'une pièce pour aller vers une autre est donc cruciale pour cette gestion.

5.3.5 La file de rayons à propager

La structure, nommée $File$ dans l'algorithme décrit précédemment, permet de stocker les rayons sortants d'une pièce pour permettre de les gérer par lots. En effet, les rayons qui sortent d'une pièce ne peuvent le faire que par les ouvertures. Plusieurs rayons vont atteindre une même ouverture pour aller dans la pièce adjacente. Le principe de regroupement permet donc de ne pas charger les meubles d'une même pièce pour chaque rayon. Les meubles d'une pièce ne doivent être chargés que lorsque l'ensemble des rayons entrant dans cette pièce est connu. Ce traitement par lots permet d'économiser le coût d'accès disque, mais pour cela la structure de stockage doit être correctement définie.

Dans notre algorithme, l'image est générée ligne à ligne et les rayons sont stockés au fur et à mesure dans la file. Les rayons sortants de la pièce et entrant dans d'autres pièces sont donc classés suivant l'ordre de parcours de l'image et non pas suivant la pièce où ils se dirigent, comme le montre la figure 5.18.a. Nous obtenons donc des groupes de rayons classés par hauteur et au

pire nous chargeons et déchargeons les pièces autant de fois qu'il y a de lignes de pixel. Ceci est le troisième problème de cet algorithme : si nous ajoutions des rayons lumineux pour calculer l'éclairage de la scène, les rayons stockés seraient complètement désordonnés.

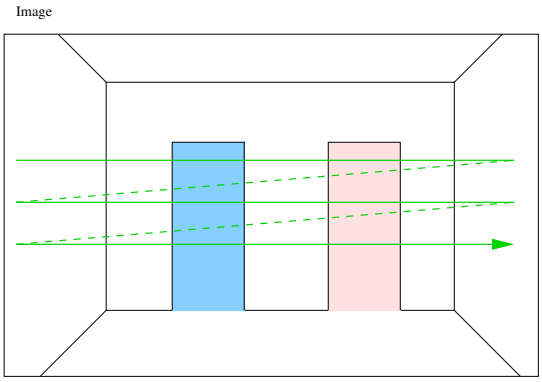


Figure 5.18 – *Les pixels de l'image étant balayée ligne par ligne, les rayons intersectant des ouvertures sont stockés dans la file en alternance : un groupe touchant une première ouverture, un groupe touchant la deuxième, puis de nouveau un groupe touchant la première, etc. Nous devons donc trier les rayons par ouverture pour limiter le nombre de chargement d'une pièce.*

5.3.6 Discussion

Nous avons développé par la suite un autre algorithme corrigeant les différents problèmes exposés précédemment. Il utilise une structure de données plus rapide d'accès, à base de faces reliées entre elles par les liaisons α_2 et α_3 seulement, et dont les informations géométriques sont directement accessibles. De plus, chaque face est triangulée, et le calcul d'intersection rayon/triangle est accéléré grâce à une grille régulière placée dans chaque pièce.

Puisqu'il est indispensable de trier les rayons par pièce, nous avons décidé d'utiliser une structure de données plus adaptée que la file pour stocker les rayons. Comme le nombre de pièces dans lequel les rayons entrent est fini (il est majoré grossièrement par le nombre de pièces), nous utilisons le tri postal (en $o(n)$ avec n le nombre d'éléments à trier). La structure de données choisie est donc un tableau de listes de rayons : une liste de rayons entrants par pièce. Cette optimisation a été mise en place dans notre algorithme final de lancer de rayons (expliqué dans la section 5.5).

Avant d'expliquer le nouvel algorithme, nous présentons une adaptation du tracé de rayons précédent. Le principe est de le réutiliser au niveau des étages pour prétraiter la visibilité œil/pièce, afin d'accélérer un lancer de rayons classique en n'effectuant le calcul que sur les pièces visibles depuis l'observateur, et non sur le bâtiment complet. Cette technique utilise la hiérarchie de cartes généralisées sans prendre en compte les meubles pour éviter les problèmes précédemment cités.

5.4 Pré-calcul de visibilité pour Pov-Ray

Puisque l'algorithme précédent de tracé de rayons donne des temps de calculs assez longs dans le cas d'un bâtiment complet, il ne peut pas directement être utilisé pour créer un lancer de rayons récursif. Nous avons néanmoins remarqué que sans traiter les meubles, les temps de calcul sont tout à fait acceptables. En effet, l'essentiel du temps passé lors des calculs est consacré à l'estimation des intersections avec les polygones des meubles. En utilisant uniquement le contour des pièces, cet algorithme permet rapidement de déterminer dans un grand bâtiment les visibilités œil/pièce, sans calculer d'autre structure de subdivision de l'espace, puisque la topologie complète du bâtiment est connue à l'aide de la carte généralisée. Nous avons choisi de modifier l'algorithme afin de prétraiter les scènes pour des algorithmes de lancer de rayons. En effet, la plupart des logiciels de lancer de rayons sont optimisés pour calculer très rapidement une image dans une scène, mais leurs structures accélératrices ne permettent pas pour autant de travailler avec des complexes architecturaux. Le nombre de faces à traiter dépasse souvent les capacités de ces logiciels. Il est fréquent d'obtenir des dépassements de mémoire et de ne pouvoir obtenir d'images de bâtiments, puisque ces derniers sont des scènes très étendues dans l'espace qui dépassent le million de polygones.

Afin de tester notre idée de pré-traitement de visibilité, nous avons décidé de travailler avec le lancer de rayons Pov-Ray. Pour cela, nous exportons un bâtiment comme un ensemble de fichiers indépendants les uns des autres. Chaque fichier correspond à une pièce différente stockée au format d'entrée de POV-Ray (fichiers d'inclusion .INC). Notre algorithme de tracé de rayons a été modifié pour déterminer, sans les meubles et pour un point de vue donné, les pièces potentiellement visibles par l'observateur. Il crée à partir de l'ensemble de ces pièces un fichier de scène (fichiers POV) en référençant les fichiers d'inclusion à utiliser. L'ensemble des pièces potentiellement visibles, notées par la suite PPV, est plus précis que les ensembles de pièces PVS (*potentially visible set*) déterminées par Teller ou par Airey (voir 2.1.4), mais il est moins général, car il doit être estimé pour chaque point de vue. Ce prétraitement rapide réduit le nombre de primitives géométriques à traiter permettant ainsi à POV-Ray de générer une image sans dépassement mémoire et avec des temps de calcul raisonnables. L'estimation des PPV est déterminée à l'aide d'une adaptation du tracé de rayons vue précédemment. Le lancer de rayons Pov-Ray utilise de son côté la technique des *bounding slabs* [KK86], pour regrouper les polygones dans des boîtes quelconques à l'aide de paires de plans parallèles (voir figure 2.3). L'association de ces deux algorithmes donne de meilleurs temps de calcul.

5.4.1 Pré-traitement de visibilité

Pour ce prétraitement, le but est d'estimer pour une image (ou pour chaque image s'il s'agit d'une animation) la liste des pièces visibles depuis le point de vue. Pour ce calcul, les meubles à l'intérieur des pièces sont ignorés. Nous nous situons dans la description la plus simple des pièces de l'étage : le troisième niveau de la hiérarchie. Ce niveau ne comporte que les murs entourant les pièces. Aucun meuble ni boîte englobante ne sont pris en compte, et le nombre de polygones à traiter est très faible. La première étape consiste à localiser la pièce dans laquelle l'observateur se situe. Ce résultat s'obtient de la même manière que dans l'algorithme de tracé

```
// Ici les ouvertures sont considérées comme de simples pièces, aucune n'étant meublée.
Trouver dans quelle pièce P_0 l'observateur se trouve ;
Pour chaque pixel de l'image faire
    P_i ← P_0 ;
    Intersection_trouve ← faux ;
    Tant que (Intersection_trouve) faire
        // Au troisième niveau de hiérarchie, seuls les murs de P_i sont présents
        Pour chaque face F de la pièce P_i faire
            Calculer l'intersection entre la face F et le rayon R ;
        Fin Pour
        Soit F_min le triangle dont l'intersection est la plus proche de l'observateur ;
        Si (F_min n'existe pas) alors
            // Seulement possible en extérieur
            Intersection_trouve ← vrai ;
        Sinon
            // Il y a eu intersection dans la pièce, donc nous la marquons.
            Marquer P_i ;
            Si (F_min est une face transparente) alors
                // Nous sortons de la pièce, nous continuons dans la pièce suivante.
                P_i ← Pièce sur laquelle donne cette ouverture ;
            Sinon
                // Une face normale a été trouvée.
                Intersection_trouve ← vrai ;
            Fin Si

        Fin Si
    Fait
Fin Pour
Sauvegarder toutes les pièces marquées dans un fichier ;
```

Figure 5.19 – *Pré-traitement de visibilité oeil/pièce.*

de rayons. La seconde étape est l'estimation des faces visibles. Étant donné que les meubles ne sont pas chargés en mémoire, la totalité du bâtiment loge en mémoire. Une version modifiée du tracé de rayons envoie les rayons à travers la pièce vide puis les autres pièces visibles de l'étage en les marquant les unes après les autres. Nous estimons ainsi l'ensemble des faces visibles à l'intérieur de l'étage. Les pièces et les ouvertures traversées sont alors stockées dans le fichier POV et forment la liste des pièces/ouvertures potentiellement visibles depuis l'observateur. Ce fichier est fourni à POV-Ray pour qu'il calcule sans prise en compte des sources lumineuses l'image finale avec les meubles à partir de ces pièces seulement. Le déroulement du calcul est décrit dans l'algorithme 5.19.

Cette technique permet de diminuer considérablement le nombre de pièces chargées par Pov-Ray. En effet, si l'observateur ne voit pas d'ouverture, Pov-Ray n'effectue son calcul que sur une pièce du bâtiment. Lorsque des ouvertures sont visibles par l'observateur, le nombre de pièces n'est pas pour autant important. En général, le nombre de pièces visibles depuis un point est toujours très petit devant le nombre total de pièces d'un bâtiment et le découpage en pièces mémorisé lors de la modélisation est plus efficace qu'un algorithme général de regroupement de faces dans des volumes englobants. Par exemple, pour le bâtiment en forme d'octogone (3 étages et 232 pièces), seulement 2 ou 3 pièces sont généralement visibles à la fois sauf dans le cas des couloirs, mais même ici le nombre de pièces ne dépasse pas la vingtaine. Le nombre de meubles pris en compte pour le calcul est donc diminué d'autant. La génération d'une image sous Pov-Ray fonctionne suivant deux grandes phases : le chargement de la scène avec la génération de la structure de *bounding slabs* et le calcul de l'image elle-même. Étant donné que le nombre de faces à traiter est essentiel dans ces deux étapes, la diminution du nombre de faces à gérer simplifie à la fois la génération de la structure et le calcul d'intersections de chaque rayon. Le gain de temps est important, essentiellement grâce à ces deux points.

5.4.2 Résultats

Les tableaux de la figure 5.20 montrent les résultats des tests de ce pré-calcul sur deux des bâtiments vus précédemment. Le premier est le bâtiment en forme de L (images 4.13). Il possède 27 pièces et est formé d'environ 300 000 polygones. Le second est le bâtiment octogonal (images 4.14), beaucoup plus grand que le précédent puisqu'il possède 232 pièces et plus de 5 millions de polygones. Précisons que le chargement de la scène par Pov-Ray ({C. P.}sur la figure) ne dépend pas de la taille de l'image, seuls notre pré-traitement {Pré.} et le lancer de rayons de Pov-Ray {L.R. P.} dépendent du nombre de pixels.

Sans optimisation, le calcul des rayons primaires pour le bâtiment en L demande en moyenne plus d'une minute sous Pov-Ray, en comptant le chargement de la scène, le calcul de la structure optimisée qu'il utilise (*bounding slabs*) et le lancer de rayon en lui-même. Nous pouvons remarquer sur les temps donnés par le tableau 5.20.a que la plus grande partie du temps de calcul dans Pov-Ray sert à charger la scène et à calculer les volumes englobants. À l'aide du pré-traitement de visibilité œil/pièce s'aidant de la topologie de la scène, nous réduisons de manière importante le nombre de pièces sur lesquelles faire le calcul (de 27 pièces à moins de 5 en général pour ce bâtiment) et d'autant le nombre de faces. Pov-Ray a donc moins de difficulté à charger la scène et à créer la structure accélératrice. Dans certains cas, le disque dur n'est même plus nécessaire pour traiter les données ne logeant plus en mémoire vive (phénomène de *swap*). Lorsque la totalité des données réside en mémoire, les temps d'accès sont meilleurs et un gain de temps important en résulte.

Le problème des pertes de temps impliquées par les échanges avec le disque est encore plus visible sur la scène octogonale. Lorsque seulement 70 pièces sur 232 étaient meublées ($1,5$ millions de polygones), avec notre machine de test (AMD 1 GHz, 512 Mo de SDRAM et $1,5$ Go d'espace tampon sur le disque pour le *swap*), le calcul durait plus de 45 minutes. Une fois, ce bâtiment entièrement modélisé, il est composé de 232 pièces quasiment toutes meublées soit plus de 5 millions de polygones. Pov-Ray ne peut alors pas charger la scène, même sur un PC plus

Image	Descriptif		Optimisation				Scène brute			Gain
	P. V.	O. V.	Pré.	C. P.	L.R. P.	Total	C. P.	L.R. P.	Total	
1. Long couloir	4	8	9 s	1 s	1 s	11 s	1 m 18	6 s	1 m 24	87 %
2. Pièce bien meublée	2	3	9 s	2 s	2 s	13 s	1 m 31	7 s	1 m 38	87 %
3. Mur (rien de visible)	1	2	5 s	1 s	1 s	7 s	49 s	3 s	52 s	86 %
4. Pièce peu meublée	1	0	3 s	1 s	1 s	5 s	50 s	3 s	53 s	90 %
5. Pièce très meublée	2	1	6 s	6 s	15 s	27 s	51 s	20 s	1 m 11	62 %
6. Vue sur façade	5	4	36 s	4 s	1 s	41 s	50 s	2 s	52 s	21 %

a.

Image	Descriptif		Optimisation				Calcul brut		
	P. V.	O. V.	Pré-traitement	C. P.	L.R. P.	Total	C. P.	L.R. P.	Total
1. Maquette d'avion	2	1	10 s	1 s	2 s	13 s	Lancer de rayon impossible, mémoire insuffisante.		
2. Hall d'entrée	3	3	12 s	4 s	2 s	18 s			
3. Bibliothèque	2	1	13 s	4 s	3 s	20 s			
4. Amphithéatre	3	2	9 s	4 s	3 s	15 s			

b.

Figure 5.20 – *a. Temps de calcul d'une image* 400 *x* 300 *pour le bâtiment en L :* 313325 *polygones. {P. V.} et {O. V.} décrivent respectivement le nombre des pièces et des ouvertures visibles par l'observateur. La colonne {Pré.} fournit le temps de notre pré-traitement pour déterminer les {P. V.} et {O. V.}. {C. P.} donne le temps de chargement de la scène par Pov-Ray. {L.R. P.} indique le temps de calcul de l'image avec Pov-Ray. - b. Temps de calcul d'une image* 400 *x* 300 *pour le bâtiment octogonal : 5 240 354 polygones. Pov-Ray n'a pas pu charger la scène brute par manque de mémoire. En effet, avec 1 500 000 polygones, 512 Mo de mémoire vive et 1,5 Go de mémoire virtuelle étaient déjà nécessaires à la construction des bounding slabs.*

puissant (Pentium IV $1,6$ GHz 768 Mo DDR et 3 Go de swap disque). Ceci explique l'absence de temps de calcul dans le tableau 5.20.b. Nous pouvons aisément estimer que les temps de calcul auraient été supérieurs aux 45 minutes précédemment cités, alors que nous obtenons moins 30 secondes avec le pré-traitement.

Les temps de calcul de notre algorithme de pré-traitement de scène sont assez variables. Dans le tableau 5.20.a, les points de vue 3, 4 et 5 donnent des résultats assez proches en temps de calcul. La première est une vue sur un mur, la seconde sur une pièce peu meublée et la troisième sur une pièce très meublée. Cette similitude s'explique simplement : pour calculer l'image du mur d'une pièce, il faut tout de même tester l'ensemble des faces de la pièce, et ces trois vues sont générées dans des pièces dont les nombres de faces sont très proches. Les vues 1 et 2 montrent que les temps de calcul changent un peu en fonction du nombre d'ouvertures visibles (cas du couloir, et de la pièce meublée avec plusieurs ouvertures). Enfin, la dernière vue montre le pire des cas pour notre algorithme : l'extérieur est considéré comme une pièce, lorsqu'un rayon sort du bâtiment, nous testons si ce rayon revient dans le bâtiment (une fenêtre donnant sur une autre façade du bâtiment), ce test est très coûteux, car l'extérieur est constitué d'un grand nombre d'ouvertures par rapport à une pièce normale ; il possède toutes les portes et les fenêtres donnant sur l'extérieur. Ce cas critique reste néanmoins en dessous du temps que Pov-Ray met pour gérer la scène seule pour des images de résolution inférieure à 640 x 480. Néanmoins, si nous augmentons la résolution, le nombre de rayons envoyés à l'extérieur du bâtiment augmente et la perte de temps devient importante. À partir d'une résolution d'image 800 x 600, notre algorithme devient plus lent que Pov-Ray dans les cas de gestion de l'extérieur. La raison de cette augmentation des temps de calcul est l'augmentation du nombre de rayons à propager et la quantité d'intersections à calculer avec l'extérieur. Nous ne sommes plus dans le cas optimal où très peu de pièces entrent en jeu et où la plupart des rayons ont une intersection avec la pièce de départ.

5.4.3 Discussion

Le mécanisme de prétraitement de visibilité œil/pièce montre l'importance de la prise en compte de la topologie de la scène. Les gains réalisés en terme de temps de calcul sont suffisamment importants pour justifier l'utilisation de telles informations dans des algorithmes de visualisation. Pour la plupart des points de vue calculés, les temps de calculs sont divisés par 6, excepté dans le cas des vues d'extérieur. Ce cas peu fréquent est aisément détectable et n'arrive que pour des bâtiments non convexes. De plus, lorsque le bâtiment devient très grand (bâtiment octogonal de 5 millions de polygones), notre prétraitement permet de réaliser les calculs sur une même machine alors que la structure de données optimisée de Pov-Ray devient très gourmande en mémoire et nécessiterait un ordinateur de plus grande capacité. Les gains obtenus grâce au découpage de la scène en pièces sont donc très intéressants étant donné la pertinence de cette subdivision de l'espace. Cette décomposition correspond plus à la réelle structure du bâtiment que celles des méthodes de calcul proposées par Airey [AB90], Teller [TS91] ou Meneveaux [MBMD98], puisqu'elle a été générée automatiquement au fur et à mesure de l'étape de modélisation. Ceci confirme les attentes que nous avions sur l'utilisation des informations de modélisation durant la phase de visualisation.

Dans cet algorithme, nous ne gérons pas les sources lumineuses. Pour chaque face non transparente, si nous traçons des rayons lumineux vers les sources pour calculer l'illumination et regardons leur chemin jusqu'aux sources sans prendre en compte les meubles, nous aurons des incohérences dans la sélection des pièces intervenant dans le calcul d'illumination. En effet, si les faces d'une pièce visible par l'observateur ne sont pas éclairées par une source lumineuse, ceci ne garantit pas qu'un meuble de cette pièce ne le soit pas (voir un exemple figure 5.21). Une solution consisterait à calculer les relations de visibilité source/pièce suivant le même principe, c'est-à-dire sans les meubles. Ce calcul indépendant du point de vue peut se faire une seule fois pour tout le bâtiment. Mais dans ce cas, le nombre de pièces intervenant dans le calcul de lancer de rayons devient trop grand. Nous ajouterions à l'ensemble des PVS l'ensemble des pièces dont la source éclaire une des pièces du PVS, ainsi que les pièces intermédiaires entre la source et la pièce.

Afin d'avoir une idée du nombre de pièces qui interviendraient dans ce calcul, nous avons utilisé le précalcul en prenant en compte les sources lumineuses. Même si cette estimation est fausse (sous-estimation des pièces intervenant dans le calcul), ceci permet de minorer le nombre de pièces nécessaires au calcul. Nous avons ainsi remarqué que le nombre de pièces augmente considérablement et que Pov-Ray ne peut plus gérer le nombre de faces obtenues par manque d'espace mémoire (malgré une sous-estimation). Par exemple, pour un couloir du bâtiment octogonal où le nombre de pièces visibles par l'observateur n'est que de 3, le nombre de pièces entrant dans le calcul d'illumination s'élève alors à 19. Ceci augmente considérablement le nombre de faces à traiter pour Pov-Ray et diminue l'intérêt de notre précalcul. Nous n'avons donc pas cherché à améliorer cet algorithme de prétraitement pour une méthode de visibilité source/pièce.

Remarquons en ce qui concerne les surfaces que nous n'avons pas géré la spécularité. Le principe est d'utiliser des rayons réfléchis en plus des rayons primaires. Le point difficile est qu'une face spéculaire présente dans un meuble ne peut dans ce cas pas être ignorée. En effet, si nous plaçons un petit miroir sur un bureau et si une ouverture est visible depuis ce miroir, comme nous ne prenons pas en compte les meubles, notre estimation des pièces visibles est fausse. De plus comme pour les sources lumineuses, l'ensemble de pièces potentiellement visibles risque alors de devenir trop grand pour pouvoir être utilisé par Pov-Ray. Nous n'avons donc non plus cherché à résoudre ce problème.

5.5 Le lancer de rayons avec structure optimisée

Au vu des résultats précédents, nous avons décidé le développement complet d'un lancer de rayons optimisé s'appuyant à la fois sur une structure accélératrice (pour accélérer les intersections rayons/faces) et sur les informations topologiques (pour conserver les avantages précédemment cités). Puisque le précalcul précédent des PPV depuis l'observateur est insuffisant pour prendre en compte les sources lumineuses, nous avons développé un algorithme de lancer de rayons complet (rayons primaires et rayons d'ombres). Ajouter au calcul les rayons secondaires (rayons d'ombres) augmente le nombre de rayons à lancer ainsi que le nombre de pièces à traiter. L'algorithme précédent sans structure accélératrice devient trop lent pour lancer un si grand nombre de rayons. Ce nouvel algorithme repose à la fois :

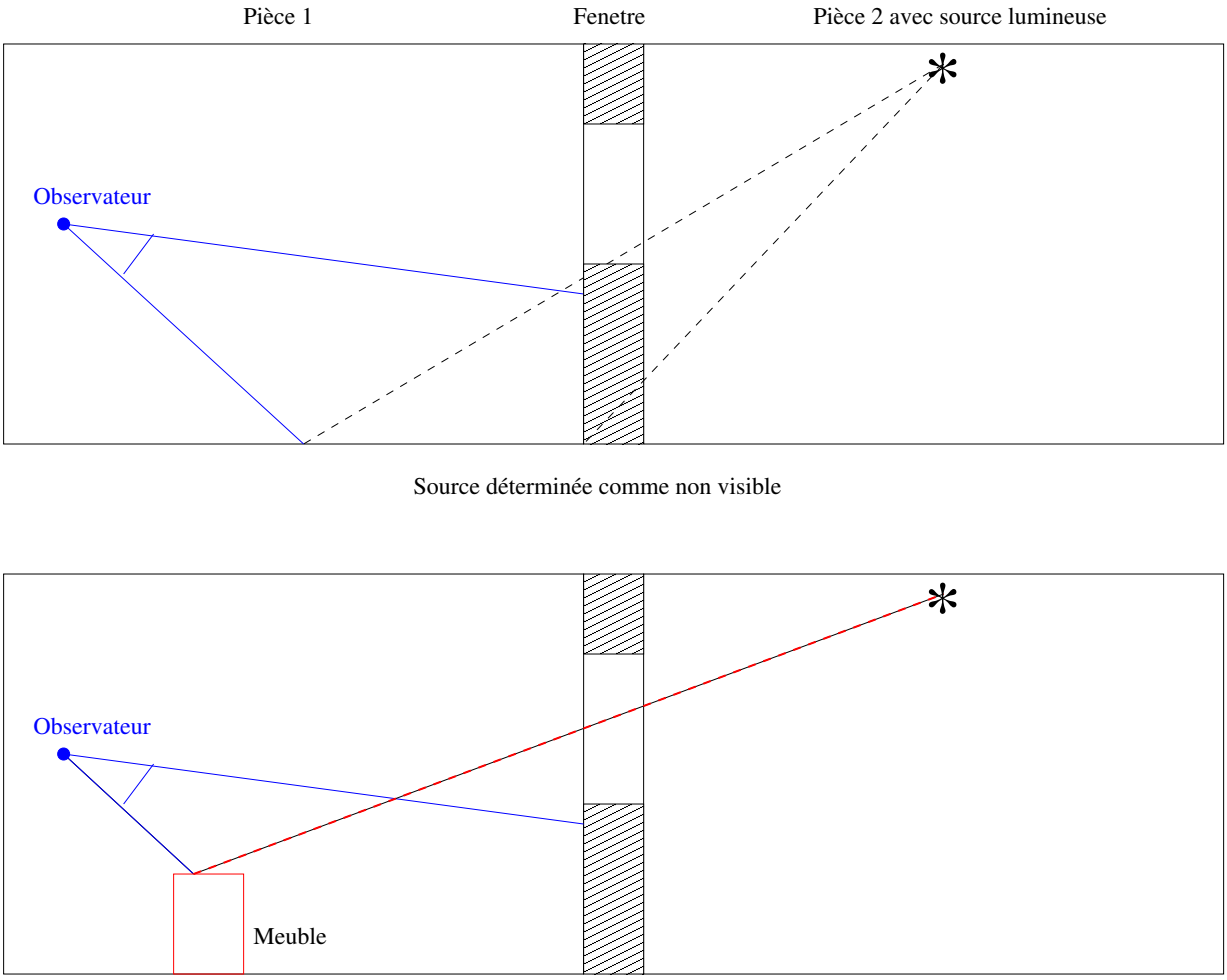


Figure 5.21 – *Les pixels de l'image étant balayée ligne par ligne, les rayons ayant une intersection avec une ouverture sont stockés dans la file en alternance : un groupe touchant une première ouverture, un groupe touchant la deuxième, puis de nouveau un groupe touchant la première, etc. Nous devons donc triés les rayons par ouverture pour limiter le nombre de chargement d'une pièce.*

- sur une structure accélératrice issue du rendu pour permettre d'accélérer le lancer de rayons dans une pièce, même avec ses meubles ;
- et sur les informations topologiques de la scène (le découpage du bâtiment) pour conserver le principe de calcul pièce après pièce en passant par les ouvertures avec gestion de rayons allant d'une pièce à l'autre.

Parmi les structures accélératrices utilisées en visualisation (présentées dans le chapitre 2.1), nous avons choisi d'utiliser les grilles régulières pour leur rapidité de construction et de parcours. Les scènes traitées sont assez régulières puisque nos pièces sont toutes du même ordre de grandeur et ont un nombre de faces souvent proches. Les grilles régulières sont à déconseiller

pour les grandes scènes ayant à un endroit particulier un très grand nombre de faces (exemple classique d'un minuscule lapin fortement maillé, au centre d'un grand stade). La grille régulière n'est pas utilisée pour contenir la totalité du bâtiment. Nous avons plutôt choisi de placer une grille par pièce afin de réduire le nombre de faces à stocker et le déséquilibre de leur disposition. Cette structure est assez bien adaptée à notre problème.

5.5.1 Principe

Nous venons de voir deux algorithmes de tracé de rayons différents utilisant directement comme structure de calcul le modèle topologique hiérarchique. L'utilisation du noyau topologique spécialisé en modélisation pour des algorithmes de visualisation ne nous permet pas de manipuler beaucoup de polygones à la fois dans un temps raisonnable, néanmoins les informations topologiques qui en sont extraites permettent d'accélérer les calculs. Sur cette base de raisonnement, nous avons choisi de développer un lancer de rayons optimisé auquel nous avons ajouté certaines informations topologiques décrivant le découpage de la scène.

Nous prenons donc comme base le premier algorithme : les calculs sont réalisés pièce par pièce en propageant les rayons. Au lieu d'avoir un modèle topologique complet, nous décrivons les pièces à l'aide d'une structure plus légère en mémoire et plus concise : une liste de faces organisées entre par les liaisons α_2 et α_3. Le découpage de la scène est représenté par différents fichiers (un par pièce) et les ouvertures servent de liens entre les fichiers. Chaque polygone est triangulé pour simplifier les calculs d'intersection avec un rayon, et chaque triangle créé connaît la face à laquelle il appartient. Nous évitons ainsi la redondance des informations de BRDF (couleur), les normales et autres paramètres en commun. Les notions de hiérarchie et de multipartition utilisées dans le modèle topologique ne sont actuellement pas prises en compte dans cet algorithme, mais elles permettent de générer le graphe d'adjacences de pièces et de rendre explicites les informations topologiques entre les faces et entre triangles. Le changement de la structure de données au centre du calcul nous donne la possibilité d'ajouter des structures accélératrices classiques en visualisation comme les grilles régulières. Ce premier changement permet de calculer beaucoup plus rapidement le tracé des rayons dans une pièce donnée. Durant le calcul, les pièces sont chaque fois chargées avec leurs meubles et les ouvertures sont placées en mémoire en même temps. Pour accélérer le calcul d'intersection et la construction de la grille régulière, toutes les faces sont triangulées.

Une seconde modification du premier algorithme a été effectuée au niveau de la structure de données gérant le passage des rayons entre deux pièces. Cette structure gérant les rayons dits *propagés* est remplacée par un tableau de listes de rayons permettant d'appliquer un tri postal par pièces. Ce tri nous permet de perdre un minimum de temps durant le calcul à classer l'information : insertion de données en $O(1)$. La structure de tableaux de files permet de stocker les rayons entrant pour chaque pièce dans une file différente.

Cet algorithme prend aussi en compte les sources lumineuses. Une fois trouvée l'intersection entre le rayon primaire et une face, nous relançons donc un rayon d'ombre vers chaque source lumineuse de la scène et, de la même manière qu'avec les rayons primaires, nous les propageons depuis la pièce intersectée jusqu'à la source de pièce en pièce à la recherche d'une intersection pouvant créer une ombre. Il s'agit de l'algorithme 5.22.

```
Soit Rayons_Piece le tableau de files de rayons ;
// Localisation
Trouver dans quelle pièce P_0 se trouve l'observateur ;
// Initialisation du tableau de files avec les rayons primaires
Pour Chaque pixel de l'image faire
    // Calcul des rayons primaires
    Créer le rayon primaire R_p allant de l'observateur au pixel ;
    Stocker dans Rayons_Piece[P_0] le rayon primaire R_p associé au pixel courant ;
Fin Pour
// Dépilement des rayons à propager : étape détaillée dans l'algorithme suivant.
// Propagation des rayons pièce après pièce
Tant que (Toutes les files du tableau Rayons_Piece ne sont pas vides) faire
    // Nous prenons les files non vides une par une
    Soit P_i une pièce dont la file est non vide ;
    // Au départ seule P_0 possède une file non vide
    Charger tous les triangles de la pièce P_i en mémoire ;
    Construire la grille régulière correspondante ;
    // Nous propageons les rayons dans cette pièce
    Pour Pour tous les rayons R à propager dans P_i faire
        Si (R est un rayon primaire) alors
            // Propagation d'un rayon primaire : voir détails dans l'algorithme suivant
        Sinon
            // Propagation d'un rayon d'ombre : voir détails deux algorithmes plus loin
        Fin Si
        Supprimer R de la file ;
    Fin Pour
Fait
// Sauvegarde de l'image
Pour chaque pixel de l'image faire
    Couleur final du pixel ← Couleur de la face * somme des illuminations ;
Fin Pour
```

Figure 5.22 – *Algorithme de lancer de rayons optimisé avec sources lumineuses : principe.*

```
// Le rayon R est un rayon primaire
Soit p le pixel associé au rayon R ;
Lancer le rayon R dans la grille régulière ;
Selon que
    Pas d'intersection :
        // Seulement possible en extérieur
        Affecter au pixel p la couleur du fond ;
    Intersection avec une ouverture :
        // Propagation du rayon primaire à la pièce adjacente
        Soit P_adjacente la pièce sur laquelle donne l'ouverture ;
        Stocker le rayon primaire R dans Rayons_Piece[P_adjacente] ;
    Intersection avec une face normale :
        Mémoriser la couleur de la face dans le pixel p ;
        // Calcul de la contribution de chaque source
        Pour Chaque source lumineuse de la scène faire
            Créer le rayon d'ombre R_o allant de l'intersection à la source ;
            Lancer le rayon d'ombre R_o dans la grille régulière ;
            Selon que
                Pas d'intersection :
                    Ajouter la contribution de cette source au pixel p ;
                Intersection avec une ouverture :
                    // Propagation du rayon d'ombre à la pièce adjacente
                    Soit P_adjacente la pièce sur laquelle donne l'ouverture ;
                    Stocker le rayon d'ombre R_o dans Rayons_Piece[P_adjacente] ;
                Intersection avec une face normale :
                    // Le pixel est dans l'ombre de cette source
                    Ne rien faire ;
            Fin Selon que
        Fin Pour
Fin Selon que
```

Figure 5.23 – *Algorithme optimisé : propagation d'un rayon primaire R dans une pièce P_i.*

Chaque file contient les rayons entrant dans une pièce donnée et le tableau de files permet de propager les rayons dans tout le bâtiment. Pour propager un rayon dans une autre pièce, un certain nombre d'informations simples sont indispensables : sa provenance, sa direction, rayon primaire ou secondaire. Les cellules des files sont donc définies de manière similaire au premier algorithme, en ajoutant quelques informations supplémentaires détaillées en 5.25.

Dans cet ensemble de données, le numéro de la pièce n'est plus stocké contrairement à l'algorithme précédent. En effet, chaque pièce possède sa propre liste de rayons entrants. De plus, un numéro permet de stocker le pixel auquel correspond le rayon, qu'il s'agisse d'un rayon primaire (pour récupérer la couleur de la face visible) ou d'un rayon secondaire (pour connaître à quel pixel ajouter l'intervention lumineuse). D'autres informations ont néanmoins été ajoutées

```
// Le rayon R est un rayon d'ombre
Soit p le pixel associé au rayon R ;
Lancer le rayon d'ombre R dans la grille régulière ;
Selon que
    Pas d'intersection :
        Ajouter cette source à la contribution lumineuse pour le pixel p ;
    Intersection avec une ouverture :
        // Propagation du rayon d'ombre à la pièce adjacente
        Soit P_adjacente la pièce sur laquelle donne l'ouverture ;
        Stocker dans Rayons_Piece[P_adjacente] le rayon d'ombre R associé au pixel p ;
    Intersection avec une face normale :
        // Le pixel est dans l'ombre de cette source : pas de contribution lumineuse
        Ne rien faire ;
Fin Selon que
```

Figure 5.24 – *Algorithme optimisé : propagation d'un rayon d'ombre R dans une pièce P_i.*

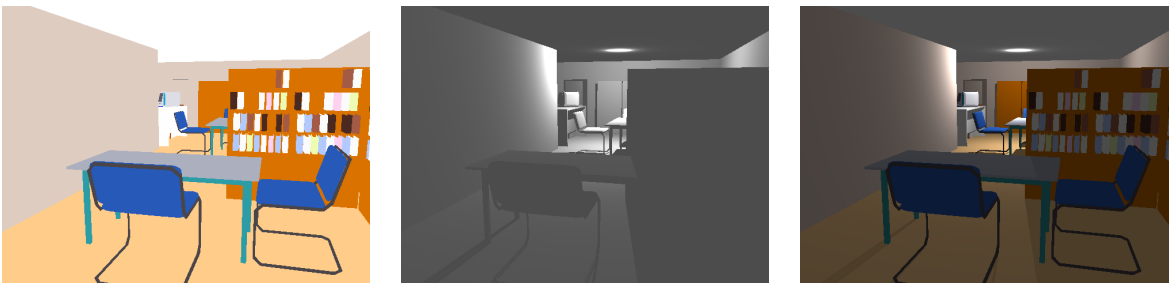
```
Struct {
    // Dernière intersection trouvée avec la scène
    Intersection : réel[3] ;
    // Point de départ du rayon
    Direction : réel[3] ;
    // Coordonnée du pixel correspondant au rayon
    Pixel : entier ;
    // numéro de la source correspondant au rayon d'ombre, -1 pour un rayon primaire
    Source : entier ;
    // Numéro de la face incidente si c'est un rayon d'ombre, -1 sinon
    Face : entier ;
    // Point de départ du rayon
    Départ : réel[3] ;
}
```

Figure 5.25 – *Structure de données permettant de stocker un rayon sortant d'une pièce pour entrer dans une autre.*

de manière à pouvoir déterminer l'éclairement apporté par la source : le numéro de la source, le numéro de la face intersectée, ainsi que le point de départ du rayon.

Pour le calcul de l'image, nous ne pouvons pas calculer directement pour chaque pixel la somme des contributions lumineuses afin d'obtenir sa couleur finale. En effet, les rayons d'ombres sont envoyés vers les sources et entrent dans d'autres pièces avant de savoir si la source intervient ou non dans le calcul. Nous devons stocker séparément la réflectance de la face et les contributions des sources. Une première image permet donc de connaître la réflectance de la face visible par chaque pixel et une seconde la somme des éclairements de chaque source, perçus par l'observateur en regardant cette face. Lorsque tous les rayons auront été tracés, nous pouvons ensuite multiplier les valeurs de chaque pixel de deux images pour obtenir le résultat final.

L'image des réflectances est remplie petit à petit lorsque les rayons primaires rencontrent des faces non transparentes de la scène. L'image des luminances est initialisée à la valeur du terme ambiant (fixé empiriquement) et les valeurs de ses pixels augmentent au fur et à mesure que les rayons d'ombres atteignent les sources sans trouver d'intersection. À la fin de l'algorithme, nous multiplions les deux images (la réflectance de la face par son illumination) pour obtenir l'image finale. La figure 5.26 montre un exemple des images de la réflectance, de la luminance et du résultat de la multiplication des deux images.



a. b. c.

Figure 5.26 – *a. Image des réflectances obtenue à l'aide des rayons primaires propagés dans la scène. - b. Image des luminances calculées à l'aide des rayons d'ombres. - c. Image finale.*

5.5.2 Résultats

Les tableaux suivants présentent les temps de calcul depuis des points de vue similaires à ceux pris pour Pov-Ray. Le premier tableau concerne le bâtiment en forme de L et le second le bâtiment octogonal. Le lancer de rayons prend en compte les sources lumineuses, soit 25 sources pour le premier et 155 pour le second. Le nombre de rayons à lancer pour chaque pixel augmente donc considérablement. Nous obtenons comme le montrent les tableaux suivants de très bons résultats par rapport à la méthode précédente sans source lumineuse. De plus, pour générer une image 640 x 480 sur le bâtiment en L avec ses lumières, Pov-Ray met dans le meilleur des cas plus de 4 minutes.

Image	P. V.	P. I.	Temps de calcul	μs/P	μs/R
1. Couloir	4	9	2 min 24	468	18
2. Pièce bien meublée	2	9	55 s	179	6,8
3. Mur	1	1	5 s	16	0,61
4. Pièce peu meublée	1	3	23 s	74	2,8
5. Pièce très meublée	2	5	1 min 16	247	9,5
6. Vue sur façade	5	11	47 s	152	5,8
7. Vue d'extérieur	11	16	1 min 22	266	10,2
8. Remise	3	8	41 s	133	5

Figure 5.27 – *Temps de calcul d'une image* 640 *x* 480 *pour le bâtiment en L :* 332141 *triangles et* 25 *sources lumineuses. {P. V.} et {P. I.} décrivent respectivement le nombre des pièces visibles par l'observateur et le nombre de pièces intervenant dans le calcul d'illumination (pièces traversées par les rayons lumineux). La colonne suivante indique le temps de calcul d'une image.* μs*/P et* μs*/R expriment les temps moyens en micro-secondes pour respectivement calculer la couleur d'un pixel et propager un rayon dans la scène (lumineux ou primaire). Les six premières images ont des points de vue similaires aux images tests de l'algorithme de pré-traitement sans sources pour Pov-Ray.*

Image	P. V.	P. I.	Temps de calcul	μs/P	μs/R
1. Maquette d'avion	2	4	3 min 25	667	4,25
2. Hall d'entrée	3	5	3 min 14	631	4
3. Bibliothèque	2	5	5 min 25	1058	6,74
4. Amphithéâtre	2	5	6 min 30	1269	8,1
5. Pièces bien meublées	3	5	9 min 41	1891	12
6. Pièce très meublée	4	7	15 min	2929	18,6
7. Pièce critique	2	5	1 h 58	23046	147

Figure 5.28 – *Temps de calcul d'une image* 640 *x* 480 *pour le bâtiment octogonal :* $5\,339\,422$ *triangles et* 155 *sources lumineuses. Les quatre premiers points de vue sont similaires aux images tests de l'algorithme de pré-traitement sans sources pour Pov-Ray.*

Pour le bâtiment en forme de L, le premier temps de calcul donné est plus élevé que les suivants. Le point de vue est dans un couloir avec plusieurs sources lumineuses visibles. Le nombre de sources lumineuses intervenant dans l'illumination n'est pourtant pas seul à l'origine de ce temps de calcul plus important, puisque les vues sur les façades du bâtiment (6 et 7 : beaucoup d'ouvertures sur des pièces avec sources) ne donnent pas des temps de calcul si grand. Le temps de calcul plus élevé est dû à la forme du couloir. Il est très long, en forme de T et traverse la totalité du bâtiment. Les triangles sont donc très longs et la grille régulière est très étalée, avec des voxels plus grands que dans les autres cas puisque nous avons fixé la résolution de notre grille. De plus, le couloir possède comme seul meuble une plante ayant un très grand nombre de petites faces : le couloir est une scène très grande et la répartition des polygones n'est pas suffisamment régulière. Ceci diminue la vitesse des calculs d'intersections et donc le temps total

de calcul. Enfin pour ce point de vue, l'extérieur intervient dans le calcul à cause des fenêtres des pièces donnant sur l'extérieur. A contrario, les vues d'extérieur prennent moins de temps malgré le grand nombre de pièces intervenant dans le calcul d'illumination, car les faces sont petites et bien réparties sur toute la façade du bâtiment. Pourtant, certains polygones de la façade posent aussi problème, car ils sont mal triangulés : ils sont longs et possèdent de nombreux sommets comme le montre la figure 5.29. En général, les vues d'intérieur sur des pièces raisonnablement meublées ou les vues depuis une fenêtre donnent de meilleurs temps de calcul (vues 2, 4, 6 et 8), souvent de l'ordre de 4 ou 5 μs par rayon.

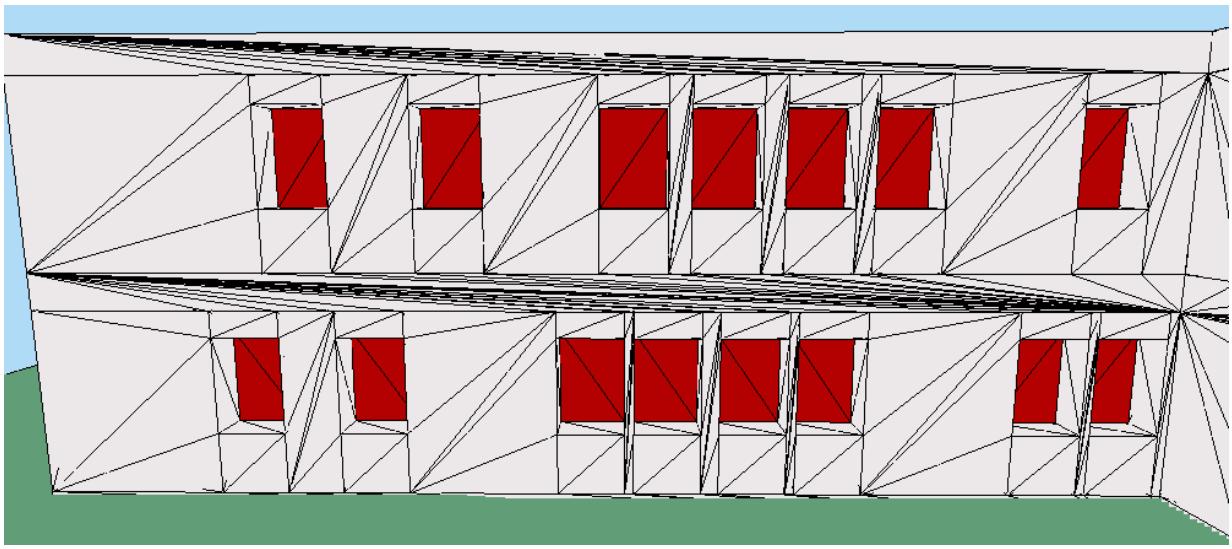


Figure 5.29 – *Exemple de problème de triangulation de faces avec de nombreux sommets.*

Pour le bâtiment octogonal, les vues proposées dans le tableau 5.28 ont toutes été réalisées en intérieur, car ce bâtiment ne possède que peu d'ouvertures sur l'extérieur. La plupart des tests effectués dans ce bâtiment montre un temps moyen de calcul de l'ordre de 6 ou 7 μs tant que la taille de la pièce reste raisonnable. Pour illustrer la faiblesse de l'utilisation d'une grille régulière par pièce, une pièce particulière a été créée avec 442733 faces (Vue 7) et prend $1,05$ Go de mémoire pour être chargée (grille régulière comprise). Le seul processus de chargement et de calcul de la grille prend déjà $9,1$ secondes. Cette pièce est le passage critique de notre algorithme puisque le *swap* disque est utilisé et ralentit tout l'algorithme. Notre optimisation repose sur le fait que le nombre de faces d'une pièce est très inférieur au nombre de faces du bâtiment, or ici la pièce représente 8% du nombre total de triangles.

5.5.3 Discussion

Les résultats obtenus sont beaucoup plus intéressants qu'avec les algorithmes précédents, vu le nombre de faces gérées et la prise en compte des sources lumineuses. L'algorithme optimisé de lancer de rayons que nous proposons permet à l'aide d'une grille régulière et de certaines

informations topologiques d'obtenir rapidement des images de grands bâtiments (quelle que soit leur taille). En effet, ici nos bâtiments font jusqu'à 5 millions de polygones et des bâtiments plus complexes peuvent potentiellement être traités avec la même efficacité. Seule la taille d'une pièce limite l'algorithme. En effet, la seule contrainte est que la place mémoire occupée par la plus grande pièce du bâtiment (structures de données comprises) ne dépasse pas l'espace mémoire disponible.

Remarquons aussi que cet algorithme n'est qu'une première utilisation des informations topologiques issues de la modélisation. Nous ne nous sommes servis ici que d'un petit nombre d'entre elles : liaisons α_2 et liaisons α_3. La hiérarchie de volumes et la multipartition n'ont pas encore été pleinement exploitées. Nous avons grâce à notre modèle topologique la possibilité de générer de nombreuses structures de données permettant d'accélérer les calculs de visualisation. En effet, la hiérarchie contient l'ensemble des informations topologiques et sémantiques essentielles à la création de structures de données plus complexes que celle que nous avons proposée. Des informations géométriques et photométriques sont de plus très simples à ajouter aux modeleurs puisqu'il ne s'agit que d'ajouter des plongements à la carte généralisée.

À l'aide de la topologie de la scène, nous pouvons aussi éviter certaines erreurs de calcul. En effet, dans un lancer de rayons classique, rien ne garantit qu'un rayon a toujours une intersection avec la scène. Il est donc impossible de différencier une erreur de calcul (un rayon passant entre deux faces par erreur) d'un résultat normal (le rayon n'atteint aucune face) et de détecter des artefacts visuels. Dans ce cas, à partir de la topologie de la scène et avec des faces transparentes sur chaque ouverture, nous savons que chaque pièce est un volume fermé par lequel il est impossible de sortir sans rencontrer une face. Quel que soit le rayon, une intersection avec la pièce (sauf pour l'extérieur) est obligatoire, soit avec une face non transparente soit avec une ouverture. Nous avons ainsi un début de correction pour notre lancer de rayons. Une erreur de calcul est donc détectée :

- lorsqu'un rayon n'a aucune intersection avec la pièce (sauf si la pièce est l'extérieur) ;
- ou lorsqu'un rayon sort de la pièce lors du parcours de la grille (simple à tester en fonction des coordonnées du pixel parcouru).

Il est possible par exemple d'utiliser une heuristique très simple pour corriger le calcul en prenant comme intersection une des deux faces dont l'arête commune est la plus proche du rayon. Néanmoins, après passage des calculs en grande précision sur les réels (8 octets pour un *double* au lieu de 4 pour un *float*), ces erreurs ne sont jamais réapparues. Nous n'avons donc pas mené cette étude plus loin. De plus, il existe d'autres types d'erreurs d'approximations des calculs que nous ne pouvons pas détecter actuellement : les erreurs d'intersection avec les meubles. Par exemple, un meuble chanfreiné avec de très petites faces pose quelquefois des problèmes comme sur l'image 5.30. Dans cet exemple, le rayon passe entre deux faces qu'il aurait dues toucher et rencontre un autre triangle à l'ombre de la source lumineuse. Si les meubles étaient construits à l'aide d'un modeleur à base topologique, les meubles formeraient eux aussi des subdivisions de l'espace. Les surfaces chanfreinées seraient stockées sous forme d'une surface maillée et un meilleur contrôle des erreurs serait alors possible pour les intersections avec un meuble.

Pour finir, nous pensons qu'il est encore possible d'accélérer les temps de calcul de notre lancer de rayons optimisé. En effet, il reste un point qui n'a pas encore été finalisé. Il s'agit du choix du format de stockage des pièces sur le disque. Ce dernier permet de réduire les coûts liés aux

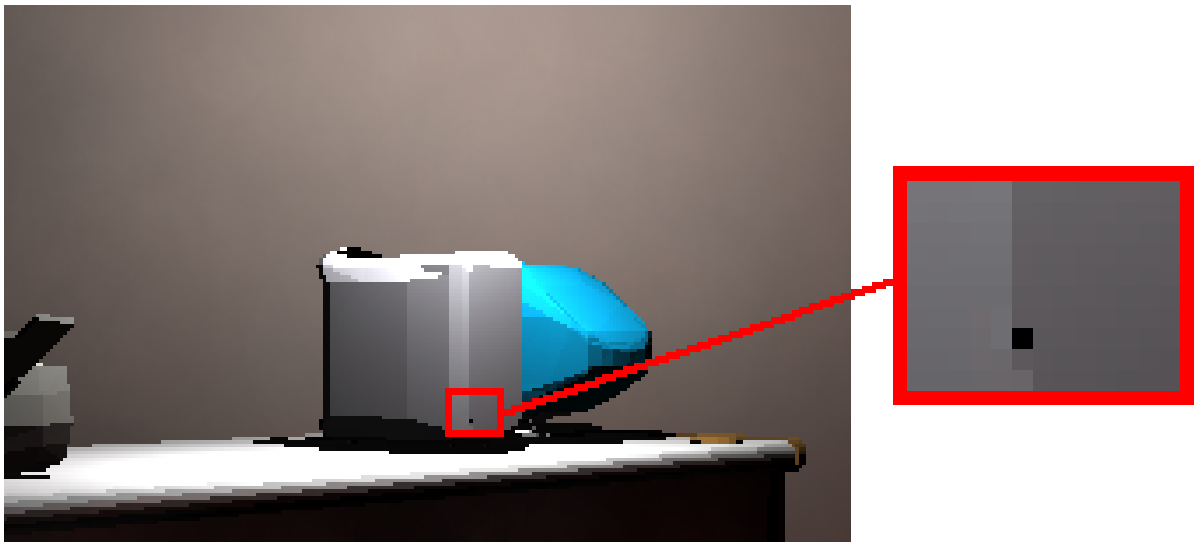


Figure 5.30 – *Exemple de problème d'approximation numérique : ici les faces de la tour de l'ordinateur sont si petites que le calcul d'intersection est faux, ce qui crée un artefact visuel.*

échanges d'entrées/sorties. Actuellement, durant le calcul, les pièces sont stockées sous forme de fichiers texte contenant à la fois la géométrie et une partie de la topologie de chaque meuble et de la pièce. Ces fichiers texte sont des fichiers NFF agrémentés d'informations topologiques. Lorsqu'une pièce est chargée, le fichier est lu puis les faces sont stockées dans la grille régulière. Ceci implique une phase d'analyse syntaxique et une phase de calcul de la grille régulière. Si les pièces étaient stockées avec leur grille régulière sous forme de fichier binaire, ces deux phases nécessitant l'utilisateur du processeur seraient supprimées. Ceci accélérerait certainement notre algorithme en diminuant les temps de chargement de chaque pièce. La grille régulière n'est peut-être pas la meilleure structure accélératrice dans notre cas. Une étude précise sur ce point et une comparaison avec les structures d'arbre BSP et de grilles hiérarchiques sera nécessaire.

CHAPITRE 6

ILLUMINATION GLOBALE PAR LANCER DE PHOTONS

Nous avons montré dans le chapitre précédent comment l'utilisation des informations topologiques et géométriques de notre modèle peut optimiser un algorithme de lancer de rayons. L'étape de modélisation fournit une subdivision de l'espace en volumes : des pièces reliées entre elles par des ouvertures. Cette décomposition aide à mieux gérer l'espace mémoire et l'accès aux données. Le lancer de rayons que nous avons développé nous a permis de générer des images de meilleure qualité qu'avec une visualisation Z-buffer. Néanmoins, nous souhaitons prendre en compte les inter-réflexions lumineuses pour plus de réalisme. Simuler ce phénomène demande beaucoup plus de calculs que pour le lancer de rayons. En effet, la difficulté est principalement due aux inter-réflexions diffuses coûteuses à calculer. Il existe de nombreuses méthodes permettant de réaliser ces calculs dans des scènes de complexité modérée : le lancer de photons, le tracé de chemins (*path tracing*) ou encore le calcul de radiosité. Pour des grands bâtiments, la méthode la plus employée repose sur le calcul de radiosité. Elle consiste à mailler chaque face de la scène et de calculer les visibilités entre chaque maille pour en déduire les échanges lumineux entre les faces. Afin de diminuer le nombre de mailles à charger en mémoire pour calculer les échanges lumineux entre une face et la scène, Airey, Teller et Meneveaux proposent de ne travailler qu'avec les pièces potentiellement visibles depuis la face traitée (*Potentially visible Set* ou PVS). La diminution du nombre de faces à traiter permet de réduire l'encombrement mémoire. Les calculs peuvent se faire soit de manière séquentielle [TFFH94, MBSB03] soit de manière parallèle [Fun96, MB99].

Le maillage de tous les polygones de la scène peut s'effectuer suivant différentes stratégies (uniforme, radiosité hiérarchique, maillage de discontinuité, etc.). Les échanges lumineux sont calculés pour toute paire de mailles de manière déterministe, ce qui implique des temps de calcul et un espace mémoire importants. Bien que cette méthode soit plus efficace en terme de précision de la simulation des inter-réflexions, nous préférons une méthode non déterministe de type Monte Carlo telle que le lancer de photons. Le stockage nécessaire ne dépend que du nombre de photons lancés depuis les sources, et pas de la complexité géométrique de la scène. De plus, cette méthode ne réduit pas le calcul des inter-réflexions aux seules surfaces diffuses, puisqu'il est généralisable à tout type de BRDF.

Notre modèle à base de cartes généralisées accepte n'importe quel type de plongement. Il est donc possible de faire évoluer notre modeleur et d'enrichir les scènes de nouvelles informations. Actuellement, en ce qui concerne les informations photométriques, les BRDF des faces et des sources lumineuses sont décrites sommairement par une couleur RVB ; les sources lumineuses sont ponctuelles et les faces sont toutes considérées comme des surfaces lambertiennes. Néanmoins, notre structure autorise n'importe quel type d'attribut et l'algorithme d'éclairement global que nous proposons sera capable d'évoluer en même temps que le modeleur. Les méthodes de calcul de radiosité sont principalement définies pour des surfaces planes lambertiennes et il est difficile à mettre en place pour d'autres types de primitives géométriques ou avec des BRDF quelconques. Nous avons préféré une technique basée sur une méthode de Monte Carlo qui permettra par la suite de simuler de nombreux effets : objets avec tout type de BRDF, caustiques, milieux participatifs, etc. De plus, les algorithmes de type Monte Carlo ne nécessitent pas de précalcul de maillage, contrairement aux méthodes de radiosité. Nous avons choisi une méthode de lancer de photons avec un stockage par carte de photons (*photon mapping*) proposée par Jensen [Jen01], car le stockage de photons est indépendant de la complexité géométrique de la scène. Enfin, comparé aux autres méthodes de Monte Carlo, le lancer de photons est moins sensible au bruit.

6.1 Principe

Le lancer de photons consiste à envoyer des photons depuis les sources lumineuses dans des directions aléatoires. Les photons sont envoyés en nombre fini dans la scène et la puissance de la source est divisée équitablement pour chacun d'entre eux. Lorsqu'un photon rencontre une face, il est stocké dans une structure de données appelée *carte de photons* (en anglais, *photon map*) puis est relancé dans la scène de manière aléatoire en ayant perdu une partie de son énergie : celle que la face a absorbée. Il existe de nombreuses heuristiques pour la mort des photons : lorsque leur énergie restante est inférieure à un certain seuil, quand un nombre maximum de rebonds est atteint, ou encore en tirant un nombre aléatoire pour savoir s'il continue à vivre. Ces traitements sont indépendants du point de vue et la création d'une image peut ensuite être réalisée soit à l'aide d'un lancer de rayons, soit par des méthodes à base de Z-buffer.

6.1.1 L'équation de luminance

Pour chaque pixel de l'image, un rayon primaire est lancé depuis l'oeil. Une fois l'intersection la plus proche de l'œil trouvée, nous connaissons le point d'intersection x dont nous devons calculons la couleur, c'est-à-dire la luminance partant de ce point dans la direction de l'observateur. Rappelons cette équation :

$$L(x, \omega_0) = L_e(x, \vec{\omega}_0) + \int_{\Omega} L_i(x, \vec{\omega}_i) f_r(x, \vec{\omega}_0, \vec{\omega}_i)(\vec{\omega}_i \cdot \vec{n}) d\omega_i \tag{6.1}$$

Comme pour le lancer de rayons, aucune face n'émet de lumière, donc L_e est nulle pour tout point x, sauf pour les sources lumineuses ponctuelles. Il reste donc à estimer correctement l'intégrale des luminances reçues. Ω est l'hémisphère représentant l'ensemble des directions d'arrivées ω_i, f_r est la BRDF de la face et L_i est la luminance incidente en un point et pour une direction donnée. La BRDF de chaque face est connue. Ici, nous nous intéressons au calcul des inter-réflexions diffuses, le modèle de BRDF choisi pour résoudre l'équation est donc celui de Lambert :

$$f_r(x, \vec{\omega}_0, \vec{\omega}_i) = f_{r,D}(x, \vec{\omega}_0, \vec{\omega}_i) \tag{6.2}$$

Jensen décompose la luminance incidente en trois termes : l'illumination directe ($L_{i,l}$), l'illumination indirecte due aux caustiques ($L_{i,c}$) et celle due aux inter-réflections diffuses ($L_{i,d}$). Les caustiques correspondent à des concentrations de lumière en certains points de la scène, en particulier par transparence derrière certaines surfaces (verre avec du liquide) ou par réflexion spéculaire (reflets lumineux sur anneau en métal).

$$L_i(x, \vec{\omega}_i) = L_{i,l}(x, \vec{\omega}_i) + L_{i,c}(x, \vec{\omega}_i) + L_{i,d}(x, \vec{\omega}_i) \tag{6.3}$$

Actuellement, nos scènes ne comportent pas d'objets générant des caustiques tels qu'une boule de cristal ou un anneau métallique. Nous posons donc $L_{i,c} = 0$. Nous ne nous intéressons qu'aux illuminations directes et aux inter-réflexions diffuses. Deux techniques différentes sont possibles pour les évaluer. La première consiste à utiliser uniquement la carte de photons pour calculer les deux intégrales. La seconde consiste à calculer les illuminations directes à l'aide d'un lancer de rayons et les inter-réflexions à l'aide de la carte de photons. Dans la suite, nous expliquons la discrétisation de l'équation pour le premier cas, la seconde méthode consistant à fusionner l'équation que nous obtiendrons avec celle du lancer de rayons.

La carte de photons contient toutes les informations nécessaires au calcul d'illumination. Ces informations sont stockées sous forme de points d'impact entre la scène et la trajectoire des photons. Chaque point d'impact est constitué d'une coordonnée $3D$, du numéro de la face intersectée, de la puissance du photon cause de l'impact et d'autres informations sur sa trajectoire. La carte de photons fournit donc des informations brutes sur le flux incident en tout point de la scène, et nous permet d'évaluer l'intégrale à partir des impacts. La luminance L_i peut s'exprimer en fonction du flux correspondant Φ_i de la manière suivante :

$$L_i(x, \vec{\omega}_i) = \frac{d^2\Phi_i(x, \vec{\omega}_i)}{(\vec{n} \cdot \vec{\omega}_i) d\omega_i dA_i} \tag{6.4}$$

Nous pouvons réécrire l'équation de luminance en fonction du flux incident de la manière suivante :

$$L(x, \omega_0) = \int_{\Omega} f_{r,D}(x, \vec{\omega}_0, \vec{\omega}_i) \frac{d^2\Phi_i(x, \vec{\omega}_i)}{d\omega_i dA_i} d\omega_i \tag{6.5}$$

Le flux incident Φ peut être estimé à l'aide de la carte de photons. Un nombre n est fixé et nous cherchons les n photons les plus proches du point x dans la carte de photons. Chaque photon transporte une puissance $\Delta\Phi_p$ et a pour direction d'incidence $\vec{\omega}_p$. L'hémisphère est donc

échantillonnée en fonction des directions d'incidence de ces n photons, et l'intégrale est alors réécrite en une somme :

$$L(x, \omega_0) = \sum_{p=1}^{n} f_{r,D}(x, \vec{\omega}_0, \vec{\omega}_p) \frac{\Delta\Phi_p}{\Delta A} \tag{6.6}$$

Le terme $\frac{\Delta\Phi_p}{\Delta A}$ correspond à une densité de photons autour du point x. Il peut être évalué à l'aide d'une technique de localisation des photons (sous-section 6.2.4).

6.1.2 Propagation des photons dans une scène simple

Le nombre de photons N à propager depuis une source lumineuse est une donnée de départ de l'algorithme. La puissance de chaque photon est de $\frac{\Phi_l}{N}$ avec Φ_l la puissance de la source lumineuse. Chaque photon doit être envoyé dans la scène selon une direction tirée aléatoirement. Pour cela, nous échantillonnons un hémisphère unitaire suivant la direction opposée au vecteur $\vec{z}$ en utilisant la formule 6.7 où r_1 et r_2 sont deux nombres aléatoires compris entre 0 et 1.

$$\begin{aligned} x &= cos(2\pi r_1)\sqrt{1 - r_2^2} \\ y &= sin(2\pi r_1)\sqrt{1 - r_2^2} \\ z &= -r_2 \end{aligned} \tag{6.7}$$

Cette équation s'obtient en échantillonnant la sphère par un ensemble de cercles à différentes hauteurs (figure 6.1). Un cercle unitaire s'échantillonne à l'aide d'un angle θ et des fonctions cosinus et sinus. Ici l'angle θ est donné par le nombre aléatoire r_1 ($\theta = 2\pi r_1$) et la hauteur h du cercle par r_2 ($h = -r_2$).

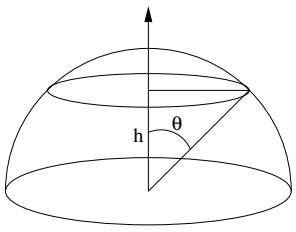


Figure 6.1 – *Échantillonner un hémisphère peut se faire en échantillonnant des cercles à différentes hauteurs.*

Une fois la direction du photon déterminée, un rayon représentant sa trajectoire est lancé dans la scène. L'intersection la plus proche est calculée et le point obtenu correspond au point x (de la formule) où le photon arrive. Le photon est alors stocké comme un photon correspondant à une illumination directe : ces photons sont appelés *photons primaires*. Le photon n'est pas entièrement absorbé par la face, il est réfléchi dans une nouvelle direction aléatoire. Un second rayon est alors lancé pour simuler un nouveau photon dont l'énergie est égale à celle du photon précédent non absorbée par la face. Le rayon rencontre alors une autre face et perd de nouveau une certaine quantité de son énergie et ce, ainsi de suite jusqu'à sa mort. Tous les photons issus

Pour chaque source lumineuse **faire**
  Soit Φ_l la puissance de la source lumineuse ;
  Soit N le nombre de photons à lancer par source ;
  Pour chaque photon à lancer **faire**
    Puissance du photon : $\Phi_p \leftarrow \Phi_l / N$;
    Déterminer une direction aléatoire $\vec{d}$ sur l'hémisphère centré sur la source ;
    Créer un rayon R_p de direction $\vec{d}$ et partant de la source ;
    Calculer P_min le point d'intersection le plus proche entre la scène et le rayon ;
    // Le photon primaire intersecte la scène
    Stocker le photon dans la photon map au point P_min comme photon primaire ;
    // Propagation dans la scène par rebond successif
    Pour chaque rebond à effectuer **faire**
      // Diminuer la puissance du photon en fonction de la BRDF f_r
      $\Phi_p \leftarrow \Phi_p * f_r * cos\theta$;
      Déterminer une direction aléatoire $\vec{d}$ sur l'hémisphère centré sur P_min ;
      Créer un rayon R_p de direction $\vec{d}$ et partant de P_min ;
      Calculer P_min le nouveau point d'intersection entre la scène et le rayon ;
      Stocker le photon dans la photon map au point P_min comme photon secondaire ;
    Fin Pour
    // Une fois les rebonds terminés, stockage de l'énergie restante dans l'ambiant
    $\Phi_p \leftarrow \Phi_p * f_r * cos\theta$;
    $Ambiant \leftarrow Ambiant + \Phi_p$;
  Fin Pour
Fin Pour
// Enfin, nous divisons l'ambiant par l'aire de la sphère
$Ambiant \leftarrow Ambiant / 2\pi$;

Figure 6.2 – *La propagation des photons avec prise en compte d'un ambiant.*

d'un rebond sont appelés *photons secondaires*. Ils forment l'illumination indirecte de la scène. La direction aléatoire dans laquelle repart le photon est déterminée avec l'équation précédente, en multipliant x, y et z par une matrice de rotation permettant d'aligner l'axe de l'hémisphère avec la normale de la face. Afin de conserver l'énergie totale de la scène, lorsqu'un photon meurt après un nombre fixé de rebonds, nous ajoutons sa puissance à un terme ambiant. À la fin du calcul, ce terme ambiant est ajouté au calcul d'illumination comme pour le lancer de rayons, après l'avoir divisé par l'aire d'un hémisphère (2π) pour exprimer le fait qu'il représente une contribution lumineuse pouvant venir de toutes les directions. L'algorithme obtenu est décrit en 6.2.

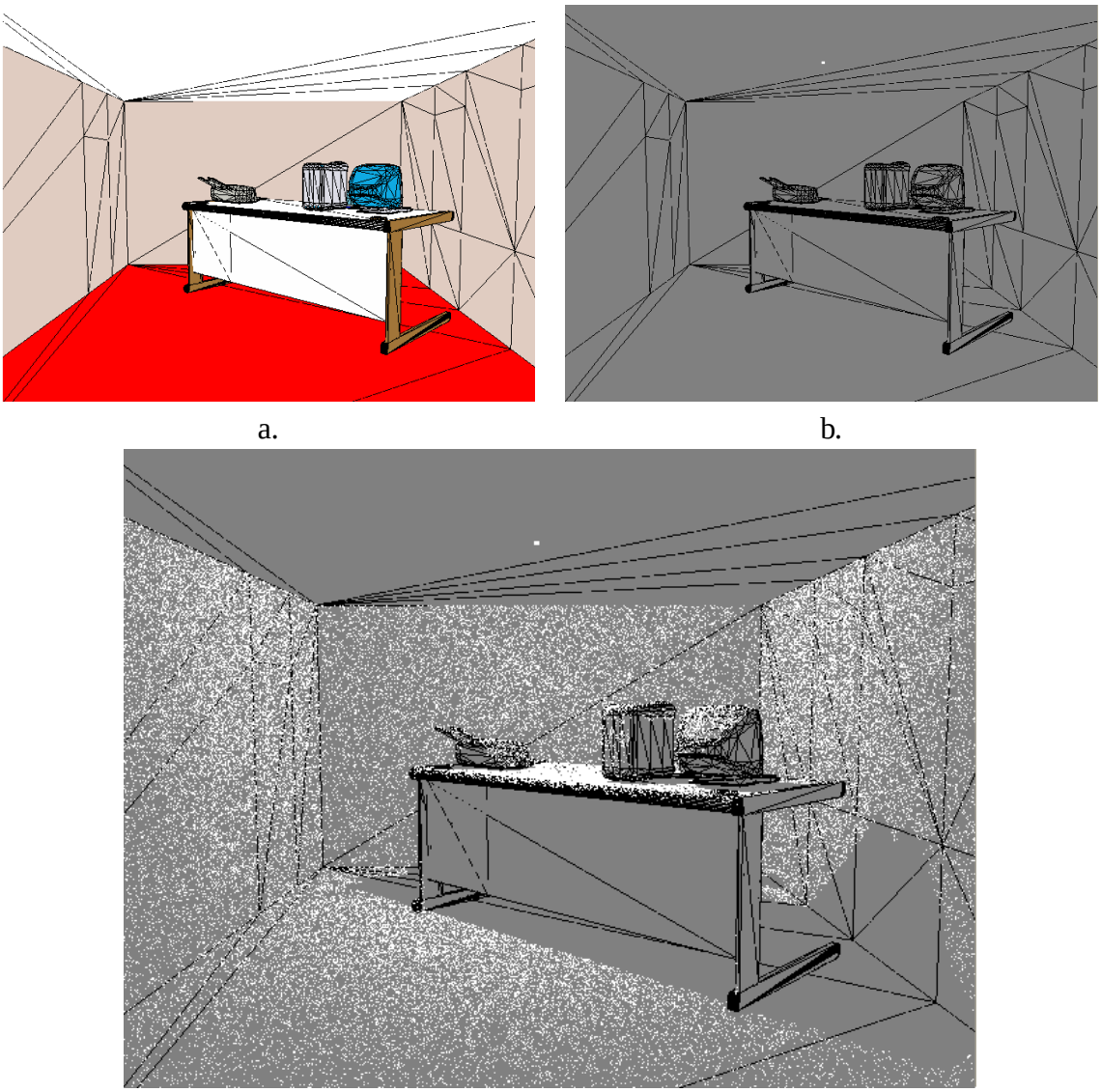
La répartition des photons dans une scène est illustrée sur les images des figures 6.3 et 6.4. Les images 6.3.a et 6.3.b montrent respectivement la scène dans laquelle sont propagés les photons et la position de la source lumineuse. Le sol de la pièce est rouge vif et le plafond est blanc. Un bureau avec un ordinateur et une imprimante se situe au milieu de la pièce. Pour cet exemple,

$100\ 000$ photons ont été lancés depuis la source lumineuse et ont rebondi 2 fois, soit un total de $300\ 000$ impacts à stocker dans la carte de photons. L'image 6.3.c montre la première étape de la propagation sans aucun rebond. Seuls les photons primaires apparaissent. Pour pouvoir les afficher, nous avons normalisé les puissances des photons entre 0 et 255. Tous les photons primaires sont blancs sur le dessin puisque la source lumineuse était blanche et aucun rebond ne s'est encore produit. Nous voyons se dessiner l'ombre du bureau sur le sol et le mur (absence de photons). Nous pouvons aussi visuellement reconnaître les zones les plus éclairées (le dessus du bureau) en repérant celles ayant les plus grandes densités de photons. Le plafond n'est pas éclairé pour l'instant, car il n'a reçu aucun photon, les rayons étant envoyés depuis la source suivant un hémisphère. Sur la dernière image (figure 6.4), l'ensemble des rebonds a été calculé. Les ombres sont atténuées par les inter-réflexions. Ces dernières se remarquent tout particulièrement sur le plafond qui est maintenant éclairé et aux bords des murs où il y a une concentration plus importante de photons. De nombreux photons de la couleur du sol (rouge) sont présents dans la scène, ce qui met en évidence les photons ayant rebondi sur le sol.

6.1.3 La carte de photons

Seuls les photons atteignant une surface diffuse sont stockés. Jensen justifie ceci en expliquant que stocker les photons arrivant sur une surface spéculaire ne donne pas d'informations utiles : la probabilité d'avoir un photon arrivant dans la direction de spécularité de la face est quasi nulle, voire nulle pour une surface spéculaire parfaite. Seul le rebond de ce type de photon est important. Nous ne nous intéressons donc qu'au stockage des photons sur les surfaces diffuses.

Une structure globale est utilisée : une carte de photons, ou plus précisément une carte des impacts de photons. Chaque photon crée plusieurs impacts tout au long de son trajet, sur chaque face diffuse qu'il a rencontrée durant ces rebonds. La particularité d'une carte de photons est de ne pas prendre en compte la géométrie de la scène dont elle est issue. Le stockage des impacts doit être très rapide pour ne pas ralentir l'étape de propagation des photons dans la scène. Les impacts sont écrits dans un tableau sans être classés pour une insertion rapide. Les photons ne sont pas uniformément répartis dans la scène, leur position dépend des sources et de la géométrie de la scène. Jensen propose ensuite de calculer un kD-tree pour mieux gérer cette répartition. La carte de photons est un ensemble de points $3D$ stockant la position des impacts sous forme d'arbre (un impact et un plan de coupe par nœud). En pratique, cet arbre est représenté par un tableau ordonné après le stockage de tous les photons. Pour simplifier et accélérer le découpage, chaque plan de coupe est perpendiculaire à un axe (x, y ou z). Cette structure permet un accès rapide aux données : pour trouver les p plus proches photons d'un point x donné dans un kD-tree de n photons, la complexité est de l'ordre de $O(p + \log n)$.



a. b.

c.

Figure 6.3 – *a. Scène triangulée dans laquelle s'effectue le lancer de photons. - b. L'emplacement de la source lumineuse est indiquée par un point blanc. Les faces sont grisés pour mieux voir les photons. Au départ, aucun photon n'est encore stocké dans la scène. - c. Au premier tour, les photons primaires ont été lancés dans la pièce. Les ombres apparaissent.*

6.2 Mise en place du lancer de photons dans de grands bâtiments

Le lancer de photons dans un grand bâtiment a été implanté suivant le même schéma que le lancer de rayons vu dans le chapitre précédent. Les photons sont propagés depuis chaque source

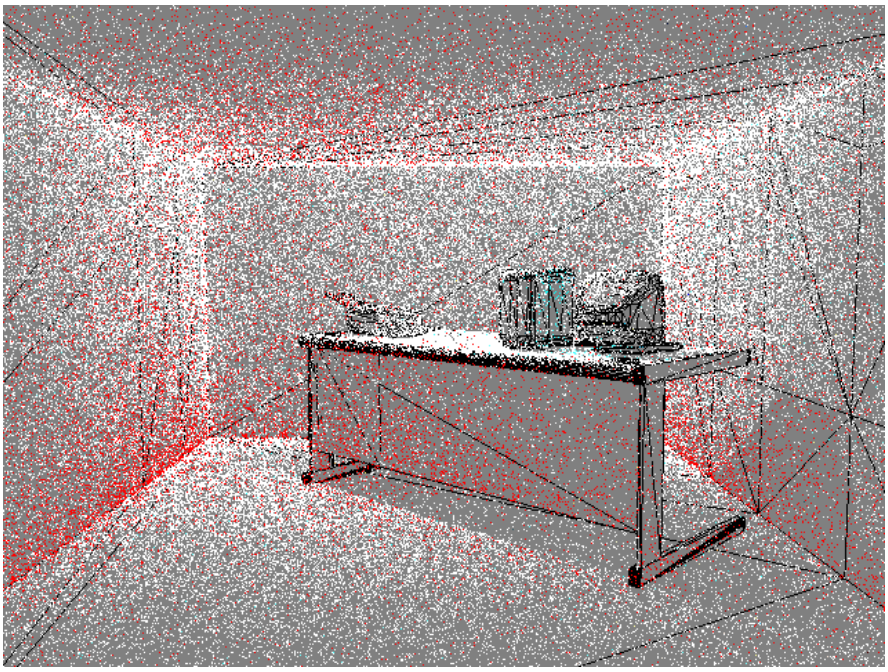


Figure 6.4 – *Après plusieurs rebonds, les photons ont été propagés dans toute la scène. les inter-réflexions sont visibles.*

lumineuse à partir de leur pièce d'origine. Une carte de photons est créée pour chaque pièce afin de stocker les impacts. Une unique carte de photons pour tous les impacts d'un bâtiment n'est pas raisonnable, car le nombre d'impacts à stocker est bien trop grand et une carte de photons contenant plusieurs millions d'impacts devient difficile à organiser en des temps raisonnables. Par exemple, pour le bâtiment octogonal possédant 232 pièces, nous avons disposé 155 sources lumineuses ponctuelles. Si au moins $50\,000$ photons par source lumineuse sont lancés, $7\,750\,000$ photons doivent alors être stockés. Ceci donne un total d'à peu près 325 Mo de données (un impact correspond à 44 octets) pour chaque rebond des photons : si un photon rebondit 3 fois sur les faces, nous obtenons $1,4$ Go de données à stocker. Créer un kD-tree à partir d'une telle quantité de données est très long, et le temps d'estimation de la luminance dans le kD-tree serait trop important pour espérer créer une image en des temps acceptables. Nous avons donc préféré utiliser une carte de photons par pièce et ainsi travailler à chaque instant avec des cartes de photons de taille plus raisonnables.

Une carte de photons est construite pour chaque pièce et la propagation des photons se traite localement, c'est-à-dire indépendamment dans chaque pièce. Des rayons représentant les trajets des photons sont envoyés dans leur pièce de départ pour trouver les intersections avec les faces. Chaque photon rencontrant une face est alors stocké dans la carte de photons correspondante. Certains photons vont durant leur trajet rencontrer une ouverture et sortir de la pièce (voir figure 6.5.a). Ces impacts avec une face transparente ne sont alors pas stockés dans la carte de photons, mais dans une structure temporaire indiquant que le photon est entré dans une autre pièce.

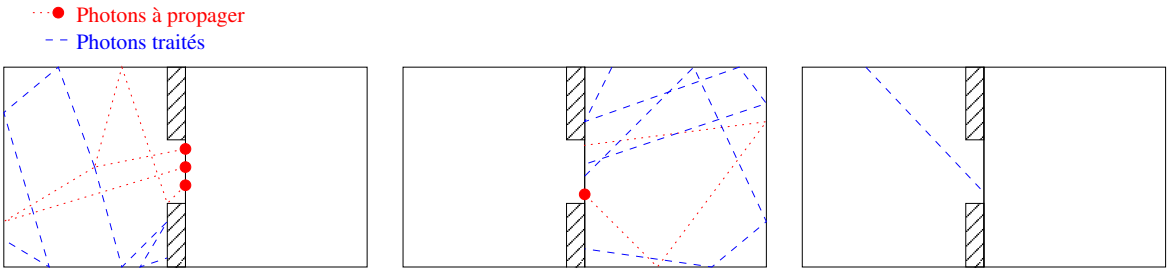


Figure 6.5 – *a. Première étape : tous les photons de la source lumineuse sont lancés dans la pièce. - b. Deuxième étape : les photons sortis par une ouverture sont propagés dansla pièce suivante. - c. Ce processus est répété jusqu'à ce qu'aucun photon ne sorte plus des pièces.*

Une fois que tous les photons de la pièce de départ ont été gérés, les photons qui ont franchi une ouverture sont propagés dans les autres pièces (figure 6.5.b). À nouveau, ces photons peuvent soit rebondir dans la nouvelle pièce et être stockés dans la carte de photons correspondante, soit en sortir et être propagés (figure 6.5.c). Plusieurs itérations sont nécessaires pour arriver à la mort de tous les photons : l'algorithme 6.2 a donc été modifié pour suivre ce principe. Lorsque tous les photons ont été propagés et stockés dans les différentes cartes de photons, l'image est générée à l'aide d'un lancer de rayons couplé à l'estimation de l'illumination indirecte : l'algorithme 6.20 est adapté pour gérer les passages par des ouvertures comme nous l'avons fait pour le lancer de rayons. Les sections suivantes détaillent le principe de nos algorithmes de propagation des photons.

6.2.1 La propagation des photons

À chaque étape, la structure temporaire utilisée pour stocker les photons atteignant des ouvertures permet un traitement par lots. Cela évite de charger et décharger chaque pièce pour un seul photon. Tous les photons entrant dans une même pièce sont regroupés pour être propagés ensemble. Ils sont donc stockés en fonction de la pièce vers laquelle ils sortent selon un tri postal. Ce principe est semblable à celui de l'algorithme du lancer de rayons proposé dans le chapitre précédent. Le principe de l'algorithme 6.2 est donc modifié pour prendre en compte les contraintes liées à la sortie d'un photon par une ouverture.

L'algorithme propage donc localement les photons pièce après pièce jusqu'à convergence. Il opère à deux niveaux : local (au niveau d'une pièce) et global (au niveau du bâtiment). Au niveau local, les photons sont envoyés depuis la source (fonction de création des photons, algorithme 6.7) puis propagés de pièce en pièce (fonction de propagation des photons, algorithme 6.8). Au niveau global, la convergence du calcul est assurée par l'algorithme 6.6. Les détails de ces trois algorithmes sont expliqués dans la suite : choix des structures de données, gestion de la mémoire, variables, etc. Dans l'algorithme 6.6, pour des raisons de conservation d'énergie, à la mort d'un photon, son énergie restante est ajoutée au terme ambiant de la pièce courante. L'ensemble des

termes ambiants est représenté par le tableau $Ambiant\_Piece$. Les photons rencontrant les faces de la scène sont stockés en partie sur le disque dur sous forme de tableaux représentant les cartes de photons de chaque pièce. Lorsque la propagation est terminée, les tableaux sont réorganisés pour donner le kD-tree correspondant. Enfin, la gestion des passages de photons par les ouvertures nécessite une structure de données permettant de stocker quels photons vont dans quelles pièces. Pour cela, la variable Tri stocke avec un tri postal les photons sortant d'une pièce comme pour le lancer de rayons avec les rayons sortants.

Nous ne nous intéressons qu'aux photons secondaires intervenant dans la simulation des inter-réflexions, nous ne stockons pas les photons primaires. L'illumination directe est calculée à l'aide du lancer de rayons exposé au chapitre précédent. Ce calcul est réalisé lors de la génération de l'image. Il n'intervient donc pas dans la propagation des photons. Le coefficient ambiant a été conservé pour stocker les photons terminant leur cycle de vie afin de ne pas avoir de perte d'énergie durant la propagation des photons. Pour stocker le terme ambiant de chaque pièce, nous utilisons un tableau de composantes RVB de taille égale au nombre de pièces. En effet, une pièce aux murs rouges et une pièce aux murs bleus ont forcément un terme ambiant (une somme d'inter-réflexions) différent. Ce tableau est petit et loge parfaitement en mémoire (700 Ko pour le bâtiment octogonal de 232 pièces et 5 millions de triangles). Néanmoins, les autres données représentant les photons prennent une place considérable pour un grand bâtiment et dépassent l'espace mémoire dont nous disposons (512 Mo de mémoire vive). Les points les plus importants de ces algorithmes sont donc la gestion de ces données en mémoire et la limitation des échanges entre la mémoire et le disque dur.

6.2.2 Gestion de la mémoire

Dans la suite, pour illustrer l'ensemble des explications, nous nous intéressons à un exemple de grand bâtiment : le bâtiment octogonal possédant 232 pièces pour un total de 5 millions de triangles. Comparé aux autres, il comporte un plus grand nombre de polygones, les pièces ont beaucoup d'ouvertures (37 ouvertures pour chacun des 4 couloirs) et la pièce la plus détaillée contient plus de $450\ 000$ triangles. Si nous souhaitons que la propagation des photons dans ce bâtiment (et tous les autres) prenne un temps minimum, nous devons réduire au maximum les accès disques et essayer de faire résider en mémoire vive le maximum de données.

Puisque nous chargeons les pièces les unes après les autres, la place mémoire prise par le bâtiment lui-même correspond au pire à la plus grande des pièces. Dans le bâtiment octogonal, il s'agit d'une grande salle de cours dont les chaises sont très détaillées. Avec plus de $450\ 000$ triangles, elle occupe en mémoire 96 Mo. En y ajoutant la grille régulière ainsi que l'ensemble des classes nécessaires à la gestion du bâtiment et des rayons à tracer, le total atteint 271 Mo (contre 251 Mo avec le logiciel de lancer de rayons). Il reste ensuite à stocker les cartes de photons et les photons en transit entre deux pièces.

Nous souhaitons lancer $50\ 000$ photons par source lumineuse. Dans notre algorithme, la mort des photons se produit après 3 rebonds ; il y a donc 4 impacts successifs avec la scène avant de mourir (nous n'en stockons que 3 puisque les photons primaires n'entrent pas dans le calcul des inter-réflexions). Notons que le nombre de rebonds peut être augmenté ou que la mort des photons peut être décidée à l'aide d'un algorithme de type "roulette russe". Néanmoins, visuel-

```
// Phase d'initialisation des données
Soit Ambiant_Piece le tableau stockant l'ambiant de chaque pièce ;
Initialisation des cases du tableau Ambiant_Piece à 0 ;
Soit Tri la structure de tri postal ;
// Initialisation du tableau Tri avec les photons venant des sources
Pour chaque pièce du bâtiment faire
    // Ici, il est inutile de charger la pièce : la position des sources suffit
    Création des photons primaires : procédure Initialisation_Photons_Primaires(...) ;
    // Initialisation des photon maps
    Création d'une photon map vide correspondant à la pièce ;
Fin Pour
// Initialisation du booléen indiquant la fin du calcul
Calcul_En_Cours ← vrai ;
// Phase de calcul
Tant que (Calcul_En_Cours) faire
    Calcul_En_Cours ← faux ;
    Pour chaque pièce du bâtiment faire
        // Le nombre de photons à propager dans une pièce est toujours présent en mémoire
        Soit N_Ph_E le nombre de photons entrant dans cette pièce ;
        Si (N_Ph_E > SEUIL) alors
            // Le nombre de photons à traiter est suffisant pour valoir le chargement de la pièce
            Chargement de la pièce en mémoire et création de sa grille régulière ;
            Propagation des photons entrants : procédure Propagation_Photons_Dans_Une_Pièce(...) ;
            Sauvegarde sur le disque dur de la photon map de la pièce ;
            Calcul_En_Cours ← vrai ;
        Sinon
            // En dessous d'un certain seuil, les photons sont ajoutés à l'ambiant
            // plutôt que propagés, le chargement de la pièce étant trop coûteux.
            Ajout de la puissance de ces photons au terme ambiant de la pièce ;
        Fin Si
    Fin Pour
Fait
// L'ensemble des photons a été propagé dans le bâtiment
// Il faut maintenant organiser le contenu des tableaux en kD-trees
Pour chaque pièce du bâtiment faire
    // Construction du kD-tree de chaque pièce
    Chargement de la photon map sauvegardée sur le disque ;
    Réorganisation des kD-tree des photons de la pièce ;
    Sauvegarde du kD-tree sur le disque à la place des morceaux de photon maps ;
    // Correction de l'ambiant de la pièce
    Division de l'ambiant de la pièce par 2π ;
Fin Pour
```

Figure 6.6 – *Au niveau du bâtiment : algorithme de propagation des photons.*

Procédure Initialisation_Photons_Primaires(...)
    Soit $P_{courante}$ la pièce traitée ;
    Pour chaque source lumineuse de la pièce **faire**
        Soit Φ_l la puissance de la source lumineuse ;
        Soit N le nombre de photons à lancer par source ;
        Pour i **de** 1 **à** N **faire**
            Poser comme point de départ du photon Ph_i la position de la source ponctuelle ;
            Fixer la puissance du photon Ph_i : $\Phi_p \leftarrow \Phi_l / N$;
            Choisir pour Ph_i une direction aléatoire $\vec{d}$ sur l'hémisphère centrée sur la source ;
            Qualifier Ph_i de photon primaire : $Ph_i.Nombre\_Rebonds \leftarrow 0$;
            Ajouter ce photon Ph_i à la case $P_{courante}$ du tableau Tri ;
        Fin Pour
    Fin Pour
Fin

Figure 6.7 – *Procédure de création des photons primaires au niveau d'une pièce (pas de propagation).*

lement dépasser plus de 3 rebonds n'améliore pas réellement la qualité de l'image et ne justifie plus les calculs supplémentaires et l'augmentation du nombre d'impacts à stocker. Nous avons donc $200\ 000$ points d'impact à mémoriser par source sous forme de photons, soit pour les 155 sources lumineuses du bâtiment octogonal un total de $1,385$ Go à stocker. Il n'est évidemment pas raisonnable d'augmenter la mémoire vive pour faire le calcul chaque fois qu'un bâtiment plus grand sera créé. Nous avons donc mis en place un système de mémorisation des photons permettant de traiter des bâtiments de n'importe quelle taille.

Les cartes de photons

Chaque carte de photons est représentée en mémoire par un tableau de photons. Les pièces étant traitées successivement, un seul tableau suffit. À l'intérieur de la pièce courante, les photons atteignant les faces n'appartenant pas à une ouverture sont stockés dans ce tableau. Pour les autres, l'impact avec une ouverture est temporairement placé dans une structure annexe (détaillée dans la section suivante) en attendant d'être propagé. Une fois la propagation terminée dans cette pièce, le tableau est sauvegardé sur le disque dur dans le fichier correspondant, comme le montre la figure 6.9. Le tableau est alors remis à zéro et une nouvelle pièce est chargée. Les anciens impacts de la nouvelle pièce n'ont pas besoin d'être lus sur le disque dur, car ils n'influent pas sur cette partie de l'algorithme. Tous les photons d'une pièce stockés sur le disque sont récupérés à la fin de la propagation pour être triés afin de former le kD-tree selon l'algorithme optimal proposé par Jensen : un tri rapide (complexité en $O(n\ log\ n)$ avec n le nombre de photons à trier).

Nous avons choisi d'allouer en mémoire au début de l'algorithme un tableau de taille suffisante pour stocker tous les impacts d'une itération dans une pièce. Nous expliquons dans les paragraphes suivants comment la taille du tableau peut être déterminée.

```
Procédure Propagation_Photons_Dans_Une_Pièce(...)
    Récupérer les photons à propager dans la pièce P_courante dans le tableau Tri ;
    Pour chaque photon Ph_i entré dans la pièce faire
        Lancer le photon Ph_i dans la pièce ;
        Soit F_min la face la plus proche intersectée par le rayon ;
        Si (F_min est une face d'ouverture) alors
            // Le photon change de pièce
            Soit P_adjacente la pièce sur laquelle donne l'ouverture ;
            Ajouter ce photon à la case P_adjacente du tableau Tri ;
        Sinon
            // La face intersectée est normale
            Si (Ph_i est un photon secondaire) alors
                // Les photons primaires se sont pas stockés dans la photon map
                Stocker le photon Ph_i dans la photon map au point d'intersection ;
            Fin Si
            // Ensuite, nous continuons à faire rebondir le photon
            Soit Ph_i.Nombre_Rebonds le nombre de fois où le photon a rebondi ;
            Pour tape de Ph_i.Nombre_Rebonds+1 à Nombre_Rebonds_A_Effectuer faire
                Calculer la puissance du photon après rebond ;
                Tirage aléatoire d'une direction sur l'hémisphère centrée au point d'intersection ;
                Lancement du photon dans la pièce P_courante ;
                Soit F_min la face intersectée ;
                Si (F_min est une face d'ouverture) alors
                    // Le photon change de pièce
                    Soit P_adjacente la pièce sur laquelle donne l'ouverture ;
                    Ajouter ce photon à la case P_adjacente du tableau Tri ;
                    Interruption de la boucle ;
                Sinon
                    // Le photon intersecte une face normale
                    Stocker le photon Ph_i dans la photon map au point d'intersection ;
                Fin Si
            Fin Pour
            Si (la boucle n'a pas été interrompue) alors
                // Le photon a donc effectué tous ses rebonds
                // Il est donc considéré comme mort et ajouté à l'ambiant de la pièce
                Augmenter l'ambiant de la pièce P_courante en lui ajoutant la puissance du photon après
                rebond sur F_min ;
            Fin Si
        Fin Si
    Fin Pour
    // Ici le tableau Tri ne contient plus de photons de la pièce P_courante
    Suppression des données temporaires de la pièce ;
Fin
```

Figure 6.8 – *Procédure de propagation des photons au niveau d'une pièce.*

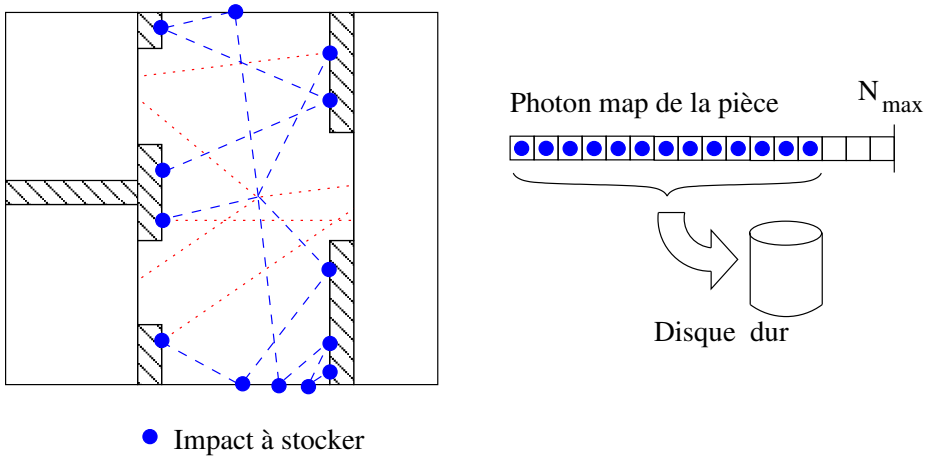


Figure 6.9 – *Le stockage des impacts se fait au fur et à mesure dans le tableau représentant la carte de photons. Seuls les impacts avec des faces non transparentes sont stockés dans la carte de photons. À la fin du calcul, le contenu du tableau est écrit sur le disque dur et le tableau est vidé pour pouvoir commencer la propagation dans une autre pièce.*

À la fin du calcul, la taille de chaque carte de photons est peu prévisible. Prenons l'exemple du bâtiment octogonal de 5 millions de polygones. Une des cartes de photons est vide : la pièce correspondante est fermée. Certaines pièces reçoivent moins de 10 photons et d'autres possèdent plus de $100\ 000$ photons. Le plus grand nombre d'impacts de photons dans une pièce est de $175\ 000$.

La taille du tableau doit donc être suffisamment importante pour contenir la plus grande carte de photons. Étant donné qu'un impact de photon ne peut disparaître, majorer le nombre total de photons à la fin du calcul nous permet de fixer une taille satisfaisante. Nous avons donc créé un tableau de N_{max} photons avec :

$$N_{max} = (Nombre\_De\_Sources\_De\_La\_Pice{+}1){*}Nombre\_De\_Photons\_Par\_Sources{*}Nombre\_De\_I \tag{6.8}$$

Nous supposons qu'une pièce ne recevra jamais plus d'impacts que ceux des photons venant de ses sources. Le $+1$ dans la formule permet de prendre en compte les photons entrants dans la pièce (interactions avec les autres pièces). Ce nombre total de photons est tout de même multiplié par 2 pour être sûr de majorer le nombre total d'impacts. Cette majoration grossière a été choisie empiriquement et n'a pas de justification théorique. Néanmoins, elle répond à deux critères vérifiés lors des tests :

- N_{max} majore le nombre total de photons à stocker dans une pièce à un instant donné. Nous avons ajouté des tests de vérification lors d'ajout de photons permettant d'alerter l'utilisateur dans le cas d'un dépassement de cette valeur. L'alerte ne s'est jamais déclenchée sur aucun bâtiment testé avec N_{max} ainsi fixé ;
- le tableau de photons de taille N_{max} ne doit pas prendre trop de place en mémoire pour ne pas gêner les calculs et le chargement des pièces. Pour le bâtiment octogonal par exemple,

$N_{max} = 300\ 000$ pour la plupart des pièces, ce qui fait un total en mémoire de 12 Mo de photons. Cette taille de tableau est tout à fait raisonnable pour la gestion d'un bâtiment complexe dont la taille des données géométriques et photométriques est de plusieurs Go.

La taille en mémoire du tableau grandit si nous augmentons le nombre de photons à lancer par sources lumineuses ou le nombre de rebonds. Il faut toujours régler correctement le nombre N_{max} pour ne pas gêner le déroulement de l'algorithme en fonction de ces paramètres. Ce réglage reste assez simple compte tenu de la taille des données gérées ici. Le stockage des photons sortant d'une pièce par une ouverture est plus difficile.

Les photons passant d'une pièce à l'autre

Puiqu'une seule pièce est chargée en mémoire à la fois, une seule carte de photons est nécessaire pour cette pièce. Néanmoins, en ce qui concerne les photons transitant entre deux pièces, nous ne pouvons pas nous contenter d'un seul tableau pour tous les photons. En effet, une pièce peut avoir plusieurs ouvertures donnant sur des pièces différentes. Lorsqu'un photon sort d'une pièce pour entrer dans une autre, le stocker dans un simple tableau sans prendre en compte sa pièce d'arrivée compliquerait la récupération du photon lors du chargement de cette nouvelle pièce. Plus précisément, si tous les photons en transit sont mélangés, un parcours supplémentaire du tableau est nécessaire pour retrouver ceux qui entrent dans la pièce que nous traitons. Ceci représente une perte de temps qui diminue les performances de l'algorithme.

La structure temporaire des photons en transit doit donc être plus élaborée. Les photons doivent être triés en fonction de la pièce où ils se dirigent pour rapidement les retrouver. Comme pour l'algorithme de lancer de rayons, nous utilisons le principe du tri postal : un tableau contenant des tableaux de photons (un pour chaque pièce). Dans l'algorithme 6.8, ce tableau de tableaux temporaires est nommé Tri. Les photons sortant de la pièce peuvent entrer dans n'importe quelle pièce adjacente, nous devons donc toujours avoir ces tableaux en mémoire pour stocker les photons en transit. La figure 6.10 nous montre l'état du tableau Tri après la phase d'initialisation (les photons des sources sont créés) et après la première propagation dans une pièce (certains photons sont sortis).

Pour chacun de ses tableaux temporaires, il nous faut définir une taille. En effet, pour le bâtiment octogonal, nous devons avoir 232 tableaux (autant que de pièces en mémoire). Si nous essayons de placer en mémoire les photons entrant dans toutes les pièces en majorant de la même façon que précédemment, la taille de la structure de données complète dépasse l'espace mémoire disponible. Sur la figure 6.11, nous avons représenté le nombre de photons entrant dans les pièces, pour chaque itération de l'algorithme. Ceci montre à chaque étape la grande disparité des quantités de données à stocker. Par exemple, au premier tour, nous pouvons clairement voir se détacher 4 pièces particulières possédant chacune plus de $70\ 000$ photons ($96\ 636$, $84\ 998$, $74\ 130$ et $83\ 964$). Ces pièces sont les couloirs des 4 étages du bâtiment. Le fait que les couloirs soient reliés à de nombreuses pièces (voir les plans de ce bâtiment sur la figure 4.14) explique le grand nombre de photons que chacun reçoit. En comparaison, les 228 autres pièces reçoivent moins de $20\ 000$ photons (soit moins d'un quart des valeurs précédentes). Il n'est donc pas raisonnable de choisir une majoration de toutes les pièces à la fois. Toujours pour ce même exemple, majorer

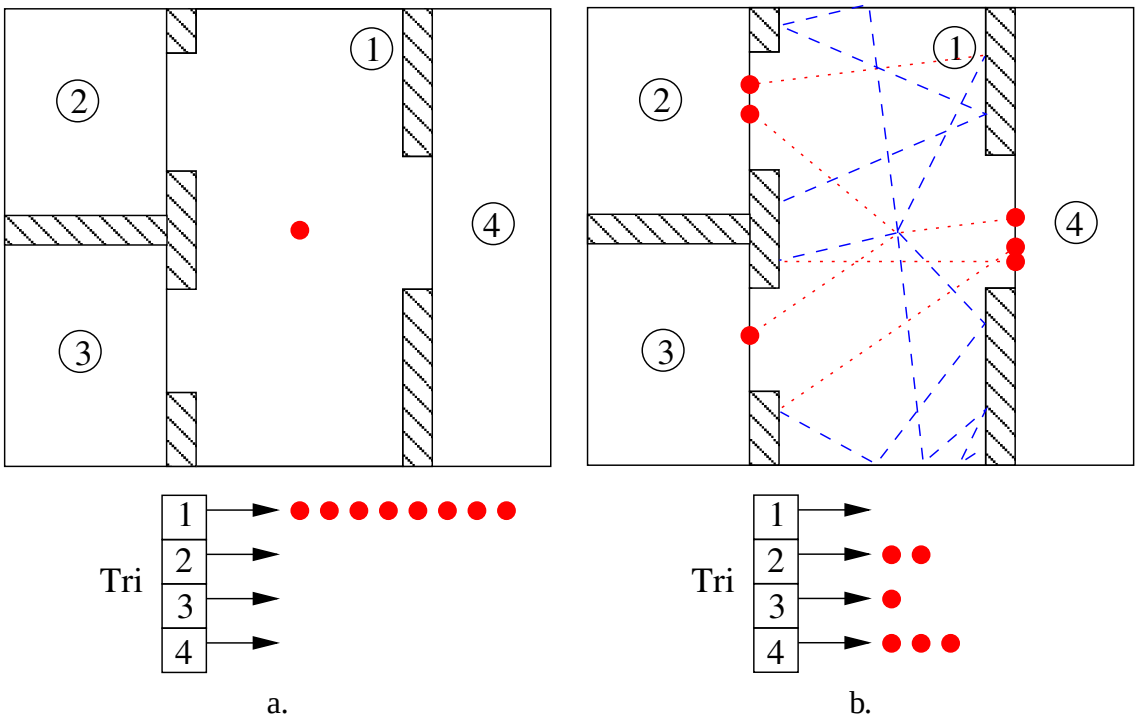


Figure 6.10 – *Exemple du stockage temporaire des photons entrant dans une autre pièce. Le bâtiment possède* 4 *pièces dont une seule a une lumière (la pièce* 1*). À gauche, après la phase d'initialisation, le tableau de la pièce possédant la source est rempli de photons tous positionnés aux coordonnées de la source. À droite, une fos la propagation des ces photons dans la pièce, quelques photons ont intersectés des ouvertures et ont été stockés dans les tableaux correspondants.*

reviendrait à choisir des tableaux de $100\ 000$ photons pour chaque pièce à cause de seulement 2% d'entre elles, ce qui donnerait une structure de données d'environ 950 Mo.

Puisqu'il n'est pas possible de stocker tous ces photons en mémoire, nous avons donc choisi de fixer une taille maximale N_{tmp} pour chaque tableau temporaire en essayant de satisfaire un maximum de pièces. Les pièces recevant plus de photons qu'il n'y a de places dans leur tableau bénéficient alors d'un fichier sur le disque dur permettant de compléter l'espace mémoire insuffisant : lorsque le tableau est plein, il est entièrement écrit sur le disque dur et vidé pour recevoir de nouveaux impacts. Afin que l'écriture sur le disque ne soit pas trop fréquente, nous fixons N_{tmp} de manière à ne le dépasser que dans peu de pièces. En regardant le premier graphique de la figure 6.11, nous pouvons à première vue fixer $N_{tmp} = 20\ 000$; de cette manière, seuls les 4 couloirs utiliseraient le disque dur. Néanmoins, ce choix n'est pas judicieux puisque dans ce cas, nous allouons un total de 194 Mo (sur seulement 512 Mo de mémoire vive) pour majorer la quasi-totalité (98%) des échanges de photons du premier tour, alors qu'au deuxième tour aucune des pièces ne dépasse $3\ 000$ photons et qu'aux tours suivants aucune ne dépasse 100 photons. Une valeur plus faible doit donc être choisie pour réduire la place mémoire et aussi se rapprocher de la taille de la plupart des échanges. Pour ce bâtiment, nous avons choisi $10\ 000$ photons,

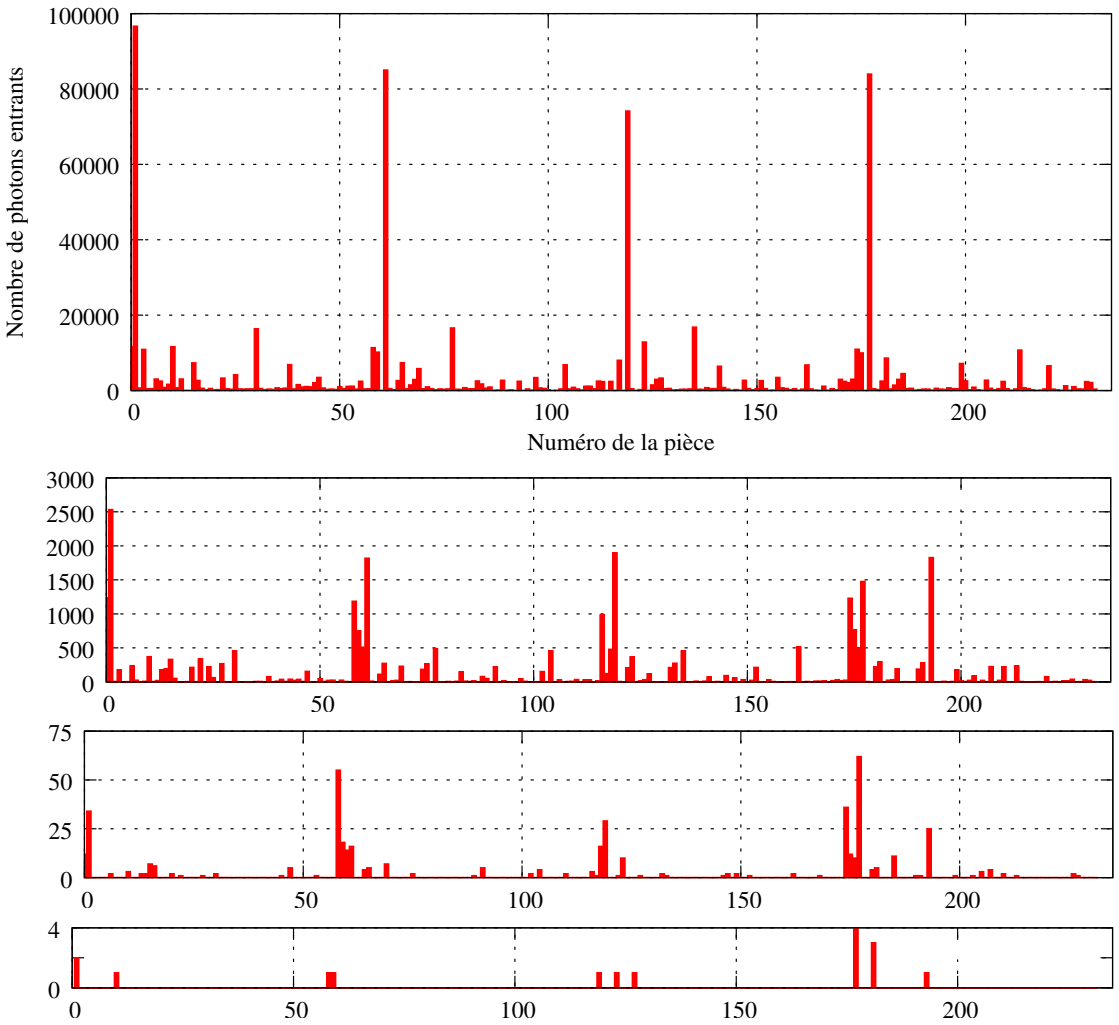
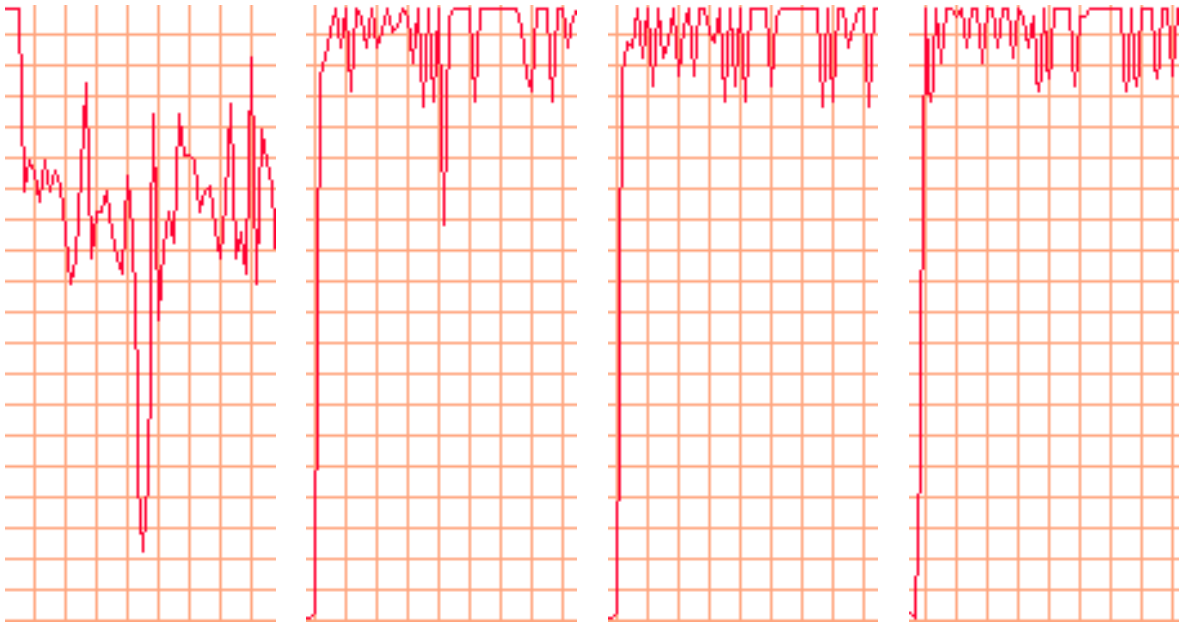


Figure 6.11 – *Histogramme du nombre de photons entrant dans une pièce à chaque étape de l'algorithme de propagation. Cet histogramme a été réalisé à partir des données de calcul obtenues sur le bâtiment octogonal. Après* 4 *étapes de propagation de photons, l'algorithme se termine.*

soit 97 Mo en tout en mémoire et seulement 15 pièces qui dépassent ce nombre d'échange (et uniquement au premier tour).

Ce nombre ne peut néanmoins pas se déterminer de cette façon avant le déroulement de l'algorithme. En effet, le raisonnement précédent ne peut se faire qu'après le calcul, or la décision se prend avant. L'explication précédente n'est ici que pour justifier la correction de ce choix. Afin de trouver la valeur de N_{tmp} sans pour autant connaître les résultats à l'avance, nous avons procédé de la manière suivante. L'algorithme est lancé durant quelques dizaines de secondes avec un nombre N_{tmp} fixé et nous regardons le temps d'utilisation du processeur durant ce laps de temps. Si la valeur choisie est trop petite, les sauvegardes sur le disque sont trop fréquentes et le processeur n'est pas suffisamment sollicité.



a. b. c. d.

Figure 6.12 – *Courbes de pourcentage d'utilisation du processeur durant le calcul (un carreau représente* 5 *secondes sur l'axe horizontal,* 5% *d'utilisation du processeur sur l'axe vertical) : a.* $N_{tmp} = 500$*, l'espace mémoire réservée étant insuffisant, les échanges avec le disque dur sont si nombreux qu'ils gênent le calcul. - b.* $N_{tmp} = 5000$*, le calcul se déroule mieux, le processus est correctement utilisé, mais quelques cas critiques arrivent encore. - c.* $N_{tmp} = 10\,000$*, le disque dur est peu sollicité et le processeur travaille fréquemment à* 100%*. Les diminutions ne sont pas dû aux tableaux temporaires, mais au chargement des pièces uniquement. d.* $N_{tmp} = 20\,000$*, par expérience, nous remarquons qu'augmenter au delà d'un certain seuil n'a pas réellement d'impact.*

La figure 6.12.a illustre le cas où le nombre N_{tmp} est très mal choisi, le pourcentage d'utilisation du processeur n'est jamais à 100% car la valeur est très en dessous de ce qui est nécessaire. Lorsque nous approchons d'un N_{tmp} correct, seules les chutes importantes de l'utilisation du processeur doivent être considérées pour nous indiquer si la valeur est mal choisie. Par exemple, sur la figure 6.12.b, une chute importante se produit dès les 40 premières secondes, le calcul durant plus de 10 minutes, cette valeur n'est donc pas à conserver. Enfin, pour de bonnes valeurs comme sur la figure 6.12.c et d, les courbes restent relativement semblables sans aucune diminution flagrante de l'utilisation du processeur. Nous choisissons alors la plus petite valeur. Ceci est une méthode évidemment empirique mais elle montre de bons résultats, car en réglant ce paramètre, nous pouvons gagner jusqu'à 1 minute de temps de calcul dans le cas du bâtiment octogonal (environ 21 minutes de calcul avec $N_{tmp} = 500$ contre 20 pour $N_{tmp} = 10\,000$). De cette manière, sur les 4 bâtiments, nous avons déterminé empiriquement qu'il faut en principe à peu près $\frac{1}{5}^{\text{e}}$ du nombre de photons lancés depuis une source.

Pour chacune des pièces, nous avons créé un tableau temporaire de photons en mémoire vive pour stocker les photons entrants. Si ces espaces tampons sont pleins, ils sont temporairement

```
// Cette fonction est appelée à chaque fois qu'un photon sort d'une pièce
// en intersectant une ouverture : stockage dans la pièce adjacente comme photon entrant.
Procédure Stocker_Photon_Entrant(Numero_Piece : entier ; Ph : Photon)
    // Nous sélectionnons le tableau temporaire de photons correspondant à la pièce
    Tableau_Temporaire ← Tri[Numero_Piece] ;
    Soit Nombre_De_Ph le nombre de valeurs contenues dans ce tableau ;
    Soit Nombre_Total_De_Ph le nombre de valeurs stockées pour cette pièce dans le tableau ou sur
    le disque ;
    // Puis nous stockons le nouveau photon
    Tableau_Temporaire[Nombre_De_Ph] ← Ph ;
    Nombre_De_Ph ← Nombre_De_Ph + 1 ;
    Nombre_Total_De_Ph ← Nombre_Total_De_Ph + 1 ;
    // Nous testons ensuite si le tableau temporaire est plein, N_tmp étant sa taille max
    Si (Nombre_De_Ph ≥ N_tmp) alors
        // Sauvegarde du tableau sur le disque
        Ouverture du fichier correspondant à la pièce Numero_Piece ;
        Copie de Tableau_Temporaire de la mémoire vers le fichier ;
        Fermeture du fichier ;
        // Remise à zéro du tableau
        Nombre_De_Ph ← 0 ;
    Fin Si
Fin
```

Figure 6.13 – *Fonction de stockage d'un photon dans le tableau Tri.*

stockés sur le disque dur. Lorsqu'une nouvelle pièce est chargée en mémoire, nous récupérons en mémoire et sur le disque l'ensemble des photons qui y sont entrés.

Dans les algorithmes 6.13 et 6.14, nous présentons les deux fonctions permettant la gestion de la mémoire au niveau des photons entrant dans les pièces. Ces fonctions manipulent les tableaux temporaires de taille N_{tmp} stockés dans le tableau Tri. La première fonction est appelée lors de la propagation quand un photon sort de la pièce courante pour entrer dans une pièce adjacente. Elle remplit les tableaux temporaires ; dès que l'un d'eux est plein, elle le vide en l'écrivant sur le disque dur. La seconde fonction est appelée lors du chargement d'une pièce pour y propager les photons entrants. Elle a pour but de réunir l'ensemble des photons entrants stockés en mémoire et sur le disque pour cette pièce.

Lors du calcul, nous avons donc en mémoire à chaque étape les triangles de la pièce traitée, sa grille régulière, sa carte de photons, ainsi que la structure temporaire de stockage des photons en transit. Pour le grand bâtiment octogonal, la pièce prenant le plus de place en mémoire (96 Mo), sa grille régulière et l'ensemble des classes nécessaires à la visualisation nécessitent au total 271 Mo. La carte de photons d'une pièce prend 12Mo et le tableau des photons en transit fait 97 Mo. Nous avons donc un total de 380 Mo de données en mémoire pour la propagation des photons à l'intérieur d'un bâtiment de 5 millions de polygones. Nous avons réussi à réduire l'espace

```
// Cette fonction est appelée lors du chargement d'une pièce afin de connaître
// l'ensemble des photons qui y sont entrés pour pouvoir les propager.
Fonction Charger_Photons_Entrants(Numero_Piece : entier) : Tableau
    Tableau_Temporaire ← Tri[Numero_Piece] ;
    Soit Nombre_De_Ph le nombre de valeurs contenues dans le tableau temporaire ;
    Soit Nombre_Total_De_Ph le nombre de valeurs stockées pour cette pièce dans le tableau ou sur
    le disque ;
    // Nous créons en mémoire un tableau pour contenir tous les photons entrant dans cette pièce
    Tableau_Complet ← nouveau Photon[Nombre_Total_De_Ph] ;
    // Nous chargeons les photons sauvegardés sur le disque
    Nombre_De_Ph_Sur_Disque ← Nombre_Total_De_Ph − Nombre_De_Ph ;
    Si (Nombre_De_Ph_Sur_Disque > 0) alors
        // Ici, nous savons que le fichier n'est pas vide donc nous le chargeons
        Ouverture du fichier correspondant à la pièce Numero_Piece ;
        Copie le contenu du fichier dans le tableau Tableau_Complet ;
        Fermeture du fichier ;
        Effacement du fichier ;
    Fin Si
    // Et nous copions le tableau temporaire de photons dans ce tableau
    Pour i de 0 à Nombre_De_Ph faire
        Tableau_Complet[Nombre_De_Ph_Sur_Disque + i] ← Tableau_Temporaire[i] ;
    Fin Pour
    Nombre_De_Ph ← 0 ;
    Nombre_Total_De_Ph ← 0 ;
    // Enfin, nous renvoyons le tableau
    Retourner Tableau_Complet ;

Fin
```

Figure 6.14 – *Fonction de chargement des photons entrant dans une pièce à l'aide du tableau Tri.*

mémoire nécessaire au calcul pour pouvoir l'exécuter avec seulement 512 Mo de mémoire vive et un système d'exploitation (Windows XP) qui occupe à lui seul 116 Mo.

Dans ces conditions de gestion mémoire, le calcul se déroule sans dépasser l'espace mémoire disponible. Pour propager les $7\ 750\ 000$ photons dans le grand bâtiment depuis les 155 sources lumineuses et équilibrer les cartes de photons, nous passons un temps total de 18 minutes.

Étude statistique

Lors du déroulement de l'algorithme, nous avons remarqué grâce à la figure 6.11 que 5 itérations (correspondants aux 4 histogrammes de photons aux ouvertures) sont nécessaires pour finir la propagation. Notons que le nombre d'itérations ne correspond pas au nombre de rebonds.

	Itération 1	Itération 2	Itération 3	Itération 4	Itération 5	kD-tree
Temps de calcul	7 min 47 s	2 min 26 s	5 min 17 s	25 s	4 s	2 min 22 s
Photons traités	7 750 000	741 954	29 581	471	36	8 522 042
% de photons traités	90,9%	8,7%	0,393%	0,0068%	0,0002%	100%
Temps par photon	60 μs	196,8 μs	10 716 μs	53 078 μs	111 111 μs	16,7 μs

Figure 6.15 – *Temps passé dans chaque itération de l'algorithme en traitant tous les photons lancés. La durée totale du calcul est de* 18 *minutes et* 21 *secondes.*

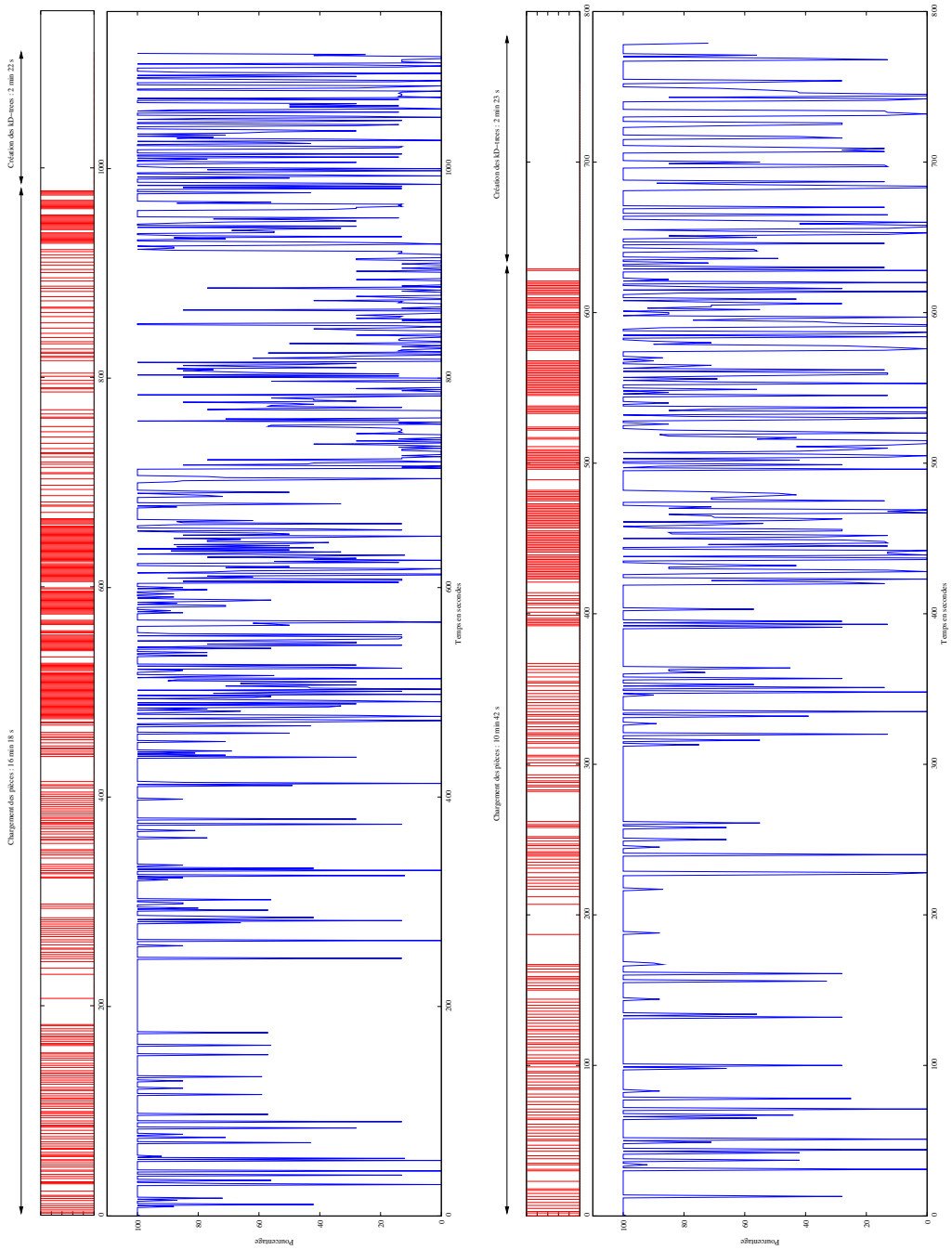
En effet, certains photons vont traverser plusieurs pièces lors d'un rebond et plusieurs itérations seront nécessaires pour trouver son impact avec le bâtiment.

Si nous mesurons le temps de calcul de chaque phase en fonction du nombre de photons gérés, nous obtenons les résultats donnés dans le tableau 6.15. Les deux premières étapes de calcul traitent plus de 99% des photons du bâtiment. Ceci signifie que les itérations suivantes sont quasiment inutiles pour estimer les inter-réflexions du bâtiment. De plus durant les dernières étapes du calcul, le temps de traitement d'un photon est de l'ordre de la milliseconde (au lieu de la microseconde auparavant). Ceci est dû au temps de chargement d'une pièce qui devient plus important que le temps de traitement des photons. Le nombre de photons à traiter dans chaque pièce devient si petit qu'il est plus long de charger la pièce que de calculer la trajectoire de ces photons. Ces problèmes sont confirmés par l'allure de l'utilisation du processeur en fonction du temps de la figure 6.16.a. Cette dernière montre clairement la phase critique (la troisième itération) où l'utilisation CPU chute considérablement à cause du changement fréquent de fichiers alors que ces derniers font quelques mégaoctets. Nous pouvons voir que l'utilisation CPU remonte en toute fin d'algorithme de propagation (avant la génération des kD-trees) puisqu'à ce moment-là les pièces chargées sont assez petites (quelques kilooctets).

Sur le graphique 6.16.a, nous pouvons aussi deviner la taille des pièces sur le disque dur : plus la pièce est grande, plus le pourcentage d'utilisation du processeur chute et plus la durée de cette diminution est longue. En effet, la mise en mémoire des faces du fichier est difficile et le processeur est moins sollicité : les fichiers de grande taille dépassent la taille de la mémoire tampon du disque dur (2 Mo) et demandent donc plus de temps de chargement.

Afin de résoudre ces problèmes, nous avons décidé pour des raisons de gain de temps de ne pas charger une pièce si le nombre de photons à traiter est trop petit. Nous avons donc fixé un seuil à ne pas franchir. Si le nombre de photons entrant dans une pièce est inférieur au seuil, nous considérons que le coût de chargement de cette pièce n'est pas justifié par le nombre de photons à propager. Un seul photon ne change pas beaucoup l'éclairement global de la pièce lorsque $50\,000$ photons ont déjà été propagés.

Ces photons sont alors ajoutés au terme ambiant de la pièce, ce qui ne nécessite aucun calcul d'intersection. Cette optimisation n'est bien sûr pas aussi juste physiquement puisque l'énergie du photon est ajoutée à la pièce dans laquelle il arrive, alors que nous ne sommes pas certains qu'il n'en sorte pas pour aller dans une autre pièce. Néanmoins, comme nous allons le voir dans



a. b.

Figure 6.16 – *Pour chacune des deux sous-figures (a et b), le petit graphique représente les accès disque (chaque barre correspond à une sauvegarde de carte de photons et un chargement de pièce) et le grand graphique montre l'évolution de l'utilisation CPU en fonction du temps : a. Le seuil fixé est nul, le calcul dure* 18 *minutes et* 31 *secondes. - b. Pour un seuil de* 50*, le calcul dure environ* 14 *minutes.*

	Itération 1	Itération 2	Itération 3	kD-tree
Temps de calcul	7 min 43 s	3 min 4 s	52 s	2 min 23 s
Photons traités	7 750 000	741 772	29 854	8 521 626
% de photons traités	90,97%	8,7%	0,32%	100%
Temps par photon	59,7 μs	248 μs	1 741 μs	16,8 μs

Figure 6.17 – *Temps passé dans chaque itération de l'algorithme pour un seuil de* 50 *photons : le temps total de calcul est* 14 *min et* 2 *sec.*

la suite, le gain en temps de calcul nous paraît justifier ce choix, en remarquant évidemment que les images finales sont très semblables.

Pour tester cette optimisation, nous avons fixé le seuil à 50 photons. Toute pièce recevant moins de 50 photons en une itération ne sera pas chargée en mémoire. Le temps de calcul est alors de 14 minutes, et si nous regardons les nombres de la figure 6.17, nous nous rendons compte que nous n'avons perdu que 416 photons dans l'ambiant. Nous avons donc sacrifié seulement $0,005\%$ de l'illumination secondaire pour gagner 4 minutes.

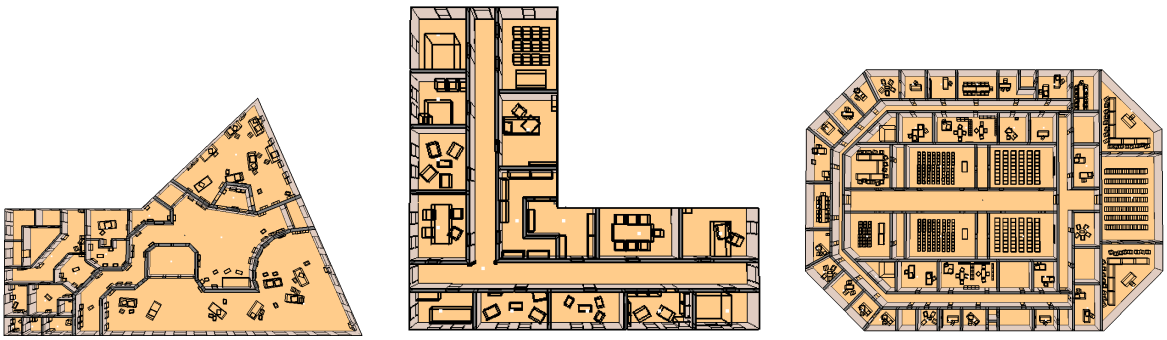
Le diagramme 6.16.b montre que le pourcentage d'utilisation CPU est fréquemment à 100% durant toute la phase de propagation et que la période où les grandes pièces étaient chargées en mémoire pour n'y propager que quelques photons n'existe plus. Ceci confirme l'intérêt du seuil : la période où peu de photons restent à propager a donc été considérablement écourtée à l'aide du seuil de photons. En dessous de cette limite, les photons ne sont pas propagés, mais ajoutés au terme ambiant de la pièce. Ceci évite de charger une pièce pour seulement 1 ou 2 photons surtout lorsque cette pièce fait plusieurs mégaoctets.

Pour estimer l'erreur de calcul générée au niveau du calcul des inter-réflexions, nous avons comparé les cartes de photons avec et sans seuil. Sans aucun seuil, le bâtiment possède une pièce ne recevant aucun photon (c'est une pièce n'ayant ni source lumineuse ni ouverture). Toutes les autres pièces du bâtiment reçoivent des photons. Avec un seuil grossier de 50 photons, nous obtenons alors 18 pièces non éclairées. Ceci peut sembler une erreur importante. Mais si nous regardons sans le seuil les cartes de photons de ces pièces, nous nous rendons compte qu'elles ne contenaient jamais plus de 60 photons et que la plupart de ces photons étaient arrivés lors de leur dernier rebond avec une énergie très faible par rapport au départ. Ceci explique pourquoi l'erreur n'est pas visible à l'œil nu sur les images. La quantité d'énergie soustraite de certaines pièces pour être mise dans le terme ambiant d'autres pièces est suffisamment petite pour ne pas être perçue.

6.2.3 Temps de calcul

Pour ce bâtiment, avec un seuil de 50 photons, la propagation de $50\ 000$ photons par source dure près de 14 minutes. À la fin du calcul, à peu près 1 Go de cartes de photons a été écrit sur le disque dur. Ces cartes de photons représentent l'ensemble des inter-réflexions calculées. L'occupation mémoire tout au long des calculs ne dépasse jamais les 512 Mo de mémoire vive puisque la mémoire utile est allouée au début du calcul à la taille nécessaire pour la plus grande

des pièces et n'augmente jamais. De plus, le pourcentage d'utilisation du processeur est très souvent au maximum. Nous avons donc un algorithme optimisé permettant d'estimer les inter-réflexions dans de grands bâtiments. Ce bâtiment possède 4 étages avec 232 pièces meublées pour un total de 5 millions de polygones. La figure 6.18.c montre le plan d'un de ses étages.



a. b. c.

Figure 6.18 – *Les plans des trois bâtiments testés : a. Plan de notre bâtiment aux formes complexes. - b. Plan d'un des deux étages du bâtiment en L. - c. Plan d'un des quatre étages de notre plus grand bâtiment, le bâtiment octogonal.*

Nous avons aussi lancé cette simulation d'éclairage sur d'autres bâtiments pour connaître les différents temps de calcul de ces derniers. Le bâtiment en L possède 2 étages avec 27 pièces pour un total de plus de 300 mille polygones. Le temps de propagation des photons suivant les mêmes paramètres que précédemment est de 1 minute et 38 secondes. Enfin, le bâtiment de formes complexes possédant 21 pièces pour un total de plus d'un million de polygones demande 3 minutes 42 secondes pour la propagation des photons.

6.2.4 Génération d'une image

Le principe est d'utiliser un lancer de rayons afin de calculer pour chaque pixel l'intersection la plus proche avec les faces de la scène. Ensuite, les photons sont utilisés pour estimer l'illumination en ce point (illumination directe et/ou indirecte selon les besoins). Pour cela, Jensen propose de rechercher les photons les plus proches du point x dans la carte de photons (figure 6.19).

La fonction $Estimation\_Eclairement$ recherche un nombre fixé de photons n à proximité du point d'intersection et en déduit une approximation de l'éclairement. Cette recherche s'effectue avec une simple comparaison de distance au point. Cette approche revient à rechercher la plus petite sphère centrée en x qui contiennent n photons. Une fois les photons localisés, il reste à estimer la densité de photons autour du point.

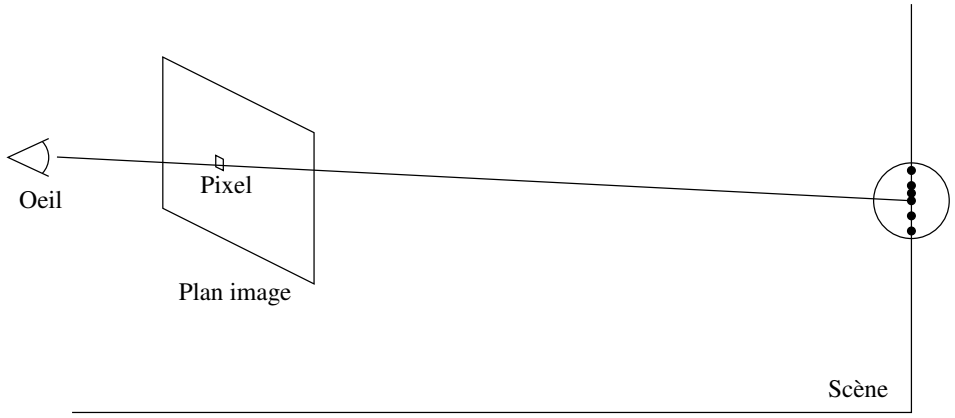


Figure 6.19 – *Un fois trouvée l'intersection entre le rayon et la scène, les photons proches du point servent à calculer l'illumination.*

Pour chaque pixel de l'image **faire**
  Créer un rayon R allant de l'observateur au centre du pixel **;**
  Calculer l'intersection la plus proche P_min entre la scène et le rayon R **;**
  // Calcul de l'illumination à l'aide de la photon map
  $I_p \leftarrow Ambiant+$ Estimation_Eclairement(P_min) **;**
  *// Calcul de la couleur du pixel : eclairement * BRDF*
  $Couleur \leftarrow I_p * f_r$ **;**
Fin Pour

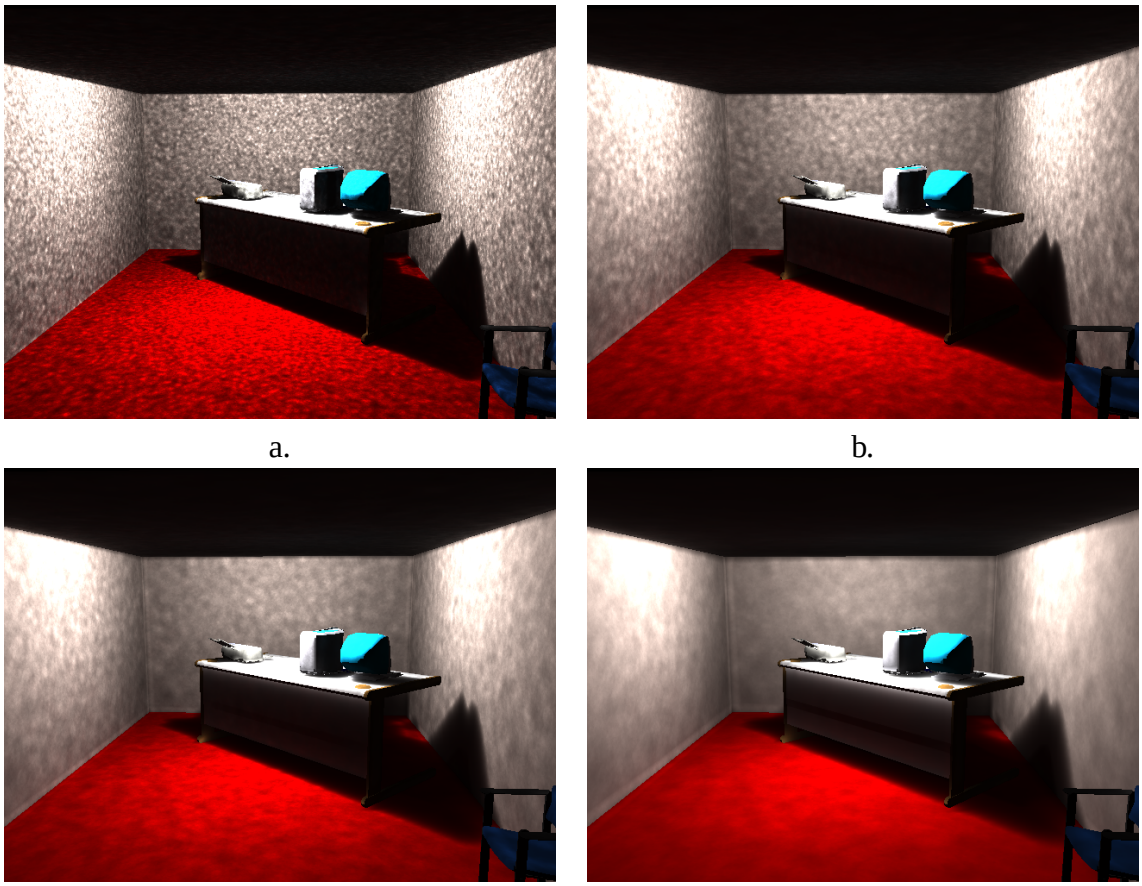
Figure 6.20 – *Génération de l'image pour des surfaces diffuses avec uniquement les photons.*

La technique la plus simple d'estimation de la densité est de sommer l'énergie des photons trouvés et de diviser cette somme par l'aire occupée par la sphère sur la surface. En posant r le rayon de la plus petite sphère contenant les n photons, l'approximation suivante donne :

$$L(x, \omega_0) \approx \frac{1}{\pi r^2} \sum_{p=1}^{n} f_{r,D}(x, \vec{\omega}_0, \vec{\omega}_p) \Delta\Phi_p \tag{6.9}$$

Les résultats de cette équation donnent des images très différentes qui dépendent fortement du nombre de photons lancés depuis la source et du nombre n de photons cherchés. Plus ces nombres sont élevés, moins l'image est bruitée. Parallèlement, les augmenter allonge les temps de calcul. En effet, augmenter le nombre de photons lancés depuis la source accroît d'autant le nombre d'impacts à stocker dans le kD-tree et ralentit la recherche. De la même façon, une augmentation de n demande la recherche d'un plus grand nombre de photons et donc un plus grand nombre de tests. Pour créer l'image de la figure 6.21, $1\ 000\ 000$ photons ont été lancés

dans une pièce et nous avons utilisé plusieurs valeurs différentes de n pour mettre en évidence son impact sur la qualité des images.



a. b. c. d.

Figure 6.21 – *a.* $n = 10$*, l'image est très bruitée mais les zones d'ombres sont plus nettes et les inter-réflexions bien visibles (bas du bureau et plafond). - b.* $n = 50$*, l'image est moins bruitée, les ombres deviennent floues mais l'éclairage du dessus du bureau est assez réaliste. - c.* $n = 100$*, le flou augmente et le bruit diminue, une erreur d'éclairement au niveau du bas du bureau devient bien visible (la partie normalement à l'ombre est illuminée). Les inter-réflexions ne sont presque plus visibles. - d.* $n = 500$*, l'erreur due à la recherche sur une sphère se voit nettement au niveau du bureau et des angles de la pièce. Les inter-réflexions sont trop atténuées.*

Ces différentes images montrent que le choix du nombre de photons est difficile puisqu'un équilibre entre le bruit et le flou doit être trouvé. Plus le nombre de photons récupérés est grand, plus la sphère s'étend et adoucit les différences d'illumination (diminution du bruit), mais plus les détails disparaissent (ombres floues, inter-réflexions atténuées) et plus les erreurs deviennent visibles (le bureau et les angles de pièces). Le premier problème est le flou au niveau des ombres, ainsi que les fréquentes variations d'éclairement sur une même face. Ceci peut se voir sur les murs des pièces et au niveau de l'ombre du bureau. L'équation simple utilisée ici ne prend pas

en compte l'éloignement de chaque photon par rapport au point, la densité de photons n'est donc pas très bien estimée. Moins le nombre de photons lancés est grand, plus le bruit est visible. Plus le nombre de photons récupérés augmente, moins les ombres sont nettes. Jensen propose donc plusieurs types de filtres consistant à affecter un poids à chaque photon en fonction de sa distance au point x. Nous pouvons aussi diminuer le bruit en utilisant des techniques issues des algorithmes de Monte Carlo : le sur-échantillonnage. Le second problème est dû à une recherche des photons sur une sphère. Sur le bord des faces, des photons ayant touché des faces différentes sont pris en compte. Plus le nombre de photons est grand, plus la sphère s'étend, plus le risque de trouver des photons n'appartenant pas à cette face est grand. Ceci crée une surestimation de l'illumination dans les coins et sur les arêtes de la pièce. Ce problème se voit surtout sur le bureau, où le bas du bureau (à l'ombre normalement) est éclairé par les photons touchant le dessus du bureau. Le schéma 6.22 illustre bien le problème. Celui-ci peut se résoudre en limitant la taille de la sphère, en ne prenant dans la sphère que les photons appartenant à la face ou encore que ceux coplanaires à cette face.

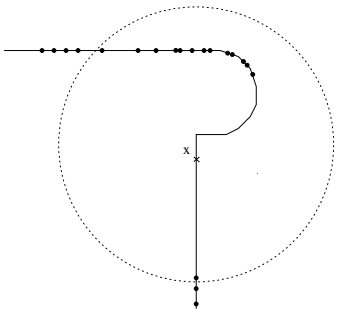


Figure 6.22 – *La recherche des n plus proches photons autour du point x sans prendre en compte la scène peut amener à récupérer des photons n'intervenant pas dans le calcul. Ici, nous schématisons la coupe du bureau des images précédentes avec un plateau chanfreiné. La source lumineuse éclaire le plateau ainsi que le bas du bureau. Le point x devrait être à l'ombre. Or des photons sont trouvés si la sphère devient trop grande.*

Les filtres

Afin d'augmenter la netteté des ombres et de prendre en compte la répartition des photons dans la sphère, les positions de ces photons par rapport au point x doivent être prises en compte. Si tous les photons sont éloignés du centre, nous pouvons être au bord d'une zone d'ombre. Diviser par l'aire du cercle n'est pas suffisant pour mettre en évidence ce déséquilibre. Jensen propose donc d'utiliser des filtres en affectant un poids à chaque photon suivant sa distance au centre x de la sphère.

$$L(x, \omega_0) \approx \frac{1}{\pi r^2} \sum_{p=1}^{n} f_{r,D}(x, \vec{\omega}_0, \vec{\omega}_p) \Delta\Phi_p w_p \tag{6.10}$$

Le premier filtre appelé filtre conique (*cone filter*) donne un poids linéairement décroissant en fonction de la distance au centre, d'où sont nom de cône. L'équation du poids est la suivante :

$$w_p = \frac{1 - \frac{d}{kr}}{1 - \frac{2}{3k}} \tag{6.11}$$

k est une constante ≥ 1 caractérisant le filtre. d est la distance entre x et le point d'impact du photon. r est le rayon de la sphère minimale. $1 - \frac{2}{3k}$ correspond à la normalisation du filtre.

Le second filtre est appelé filtre gaussien (*gaussian filter*) et le poids vaut :

$$w_p = \alpha \left[1 - \frac{1 - e^{-\beta \frac{d^2}{2r^2}}}{1 - e^{-\beta}} \right] \tag{6.12}$$

Ici deux constantes α et β caractérisent la gaussienne. La gaussienne est un filtre normalisé.

Nous avons implanté ces deux filtres afin de mieux comprendre leur intérêt. Nous avons aussi implanté une autre gaussienne assez classique dont l'intégrale vaut 1 :

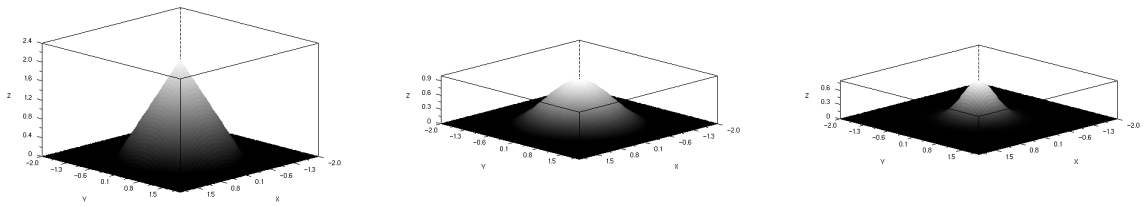
$$w_p = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{d^2}{\sigma^2}} \tag{6.13}$$

La figure 6.24 montre quatre images permettant de comparer ces trois filtres avec l'image sans filtre. Le nombre de photons lancés est le même que précédemment et n vaut 100. Les courbes en niveau de gris des trois filtres sont visibles sur la figure 6.23. Nous remarquons tout d'abord que les ombres sont mieux détourées sur les images avec filtres. Le filtre conique améliore la netteté des ombres et lisse un peu le bruit. Mais ceci n'est pas encore suffisant. Les filtres gaussiens n'aident pas plus à la diminution du bruit sur les grandes faces. En effet, Jensen remarque qu'ils sont surtout adaptés pour filtrer les caustiques, c'est-à-dire améliorer le contraste entre des zones de fortes différences lumineuses. Les filtres permettent aussi de réduire les erreurs dues à la recherche sur une sphère, puisque seuls les photons proches ont réellement une importance. Sur le bas du bureau, l'ombre est mieux définie que sans les filtres puisque les photons lointains ont moins d'importance.

Illumination indirecte seulement

Il est difficile de faire disparaître le bruit en travaillant avec une carte de photons. Ce bruit est surtout gênant sur les grands polygones et pour l'illumination directe, c'est-à-dire au niveau des ombres et des dégradés d'éclairage des grandes faces diffuses. Jensen propose donc de n'utiliser les photons que pour calculer l'illumination indirecte et les caustiques. L'illumination directe est alors calculée à l'aide d'un lancer de rayons. Cette technique permet d'avoir des faces diffuses illuminées proprement et des ombres nettes grâce au lancer de rayons, ainsi qu'une bonne estimation des inter-réflexions à l'aide des photons.

La figure 6.26 montre le résultat de l'algorithme mixte. Les ombres sont rendues plus nettes par le lancer de rayons, mais sont par endroits atténuées par les inter-réflexions. Ces dernières



a. b. c.

Figure 6.23 – *a. Courbe du filtre conique utilisé par Jensen. b. Courbe du filtre gaussien utilisé par Jensen. c. Courbe du filtre gaussien normalisé.*

peuvent aussi se voir sur le plafond qui est éclairé, le bureau qui reflète la couleur rouge du sol. L'image est d'ores et déjà beaucoup moins bruitée et plus réaliste que les images précédentes. Néanmoins, dans les zones où la plupart de l'éclairement est dû aux inter-réflexions, le bruit est encore visible. Afin d'adoucir les grandes différences entre deux pixels voisins, nous pouvons utiliser le sur-échantillonnage, c'est-à-dire lancer plusieurs rayons par pixel et calculer une moyenne des luminances obtenues.

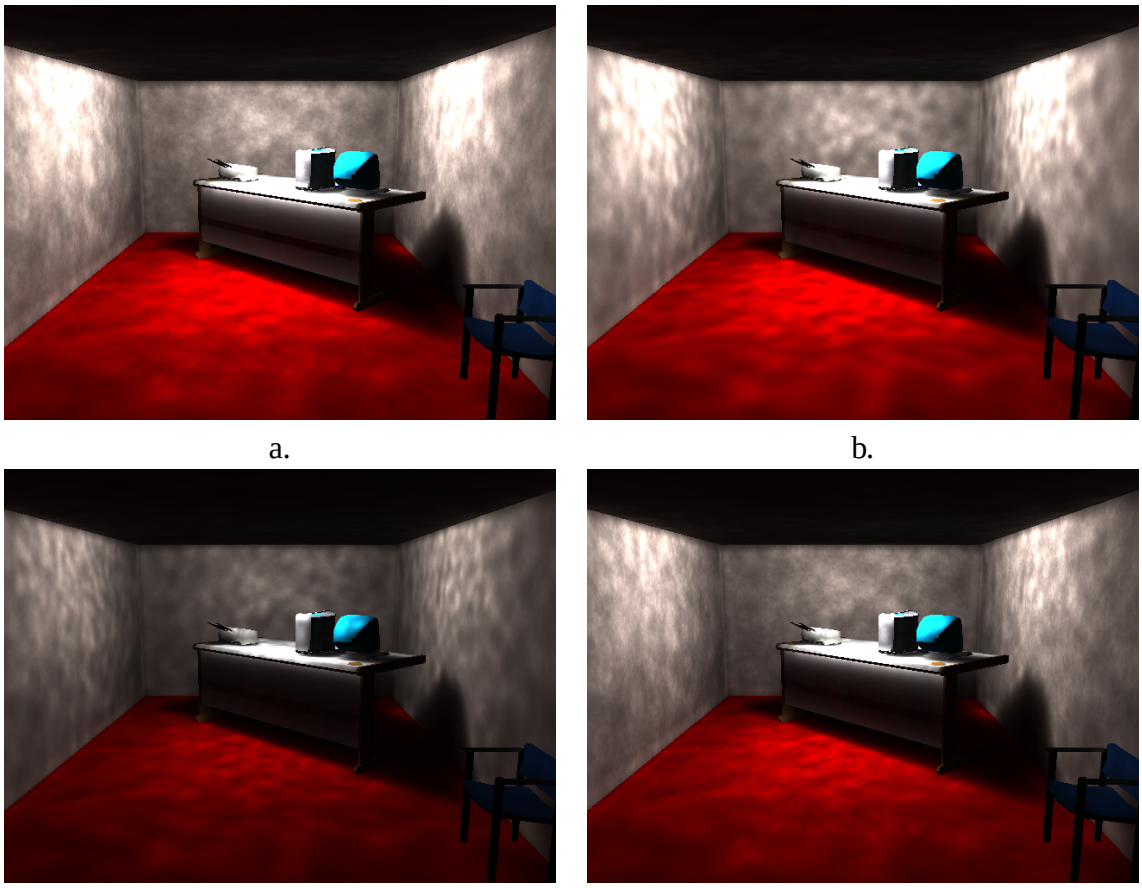
Sur-échantillonage

Le sur-échantillonage est une technique très utilisée pour supprimer le bruit dans les rendus de type Monte-Carlo. Le principe est d'échantillonner les pixels par plusieurs rayons et de faire la moyenne des luminances des échantillons pour l'affecter au pixel. De cette manière, les différences entre chaque pixel sont moins grandes et l'image paraît plus lisse, plus réaliste.

Les images obtenues sont ainsi moins bruitées, les transitions moins fortes entre les zones avec une forte densité de photons secondaires. La figure 6.28 montre deux images de nos bâtiments utilisant un sur-échantillonage des pixels : 9 rayons par pixel. La qualité des images obtenues est ainsi bien supérieure à celle des précédentes.

6.3 Discussion

L'ensemble des travaux sur le lancer de photons a été soumis à la conférence Graphics Interface en novembre 2004 [FM]. Cet algorithme nous permet donc en moins de 14 minutes d'avoir une bonne estimation des inter-réflexions dans un grand bâtiment de 5 millions de triangles. Une fois, ces transferts lumineux obtenus, nous pouvons créer autant d'images que nous le souhaitons à l'intérieur de ce bâtiment. En effet, les cartes de photons sont stockées sur le disque dur, il suffit alors d'utiliser le lancer de rayons comme expliqué précédemment. Le travail effectué sur la gestion de la mémoire nous a permis d'obtenir un algorithme ne dépendant plus de la taille du bâtiment étudié, mais de celle de sa plus grande pièce et du nombre de ces sources. Nous pensons qu'il peut être aisément adapté à des bâtiments encore plus grands.



a. b.

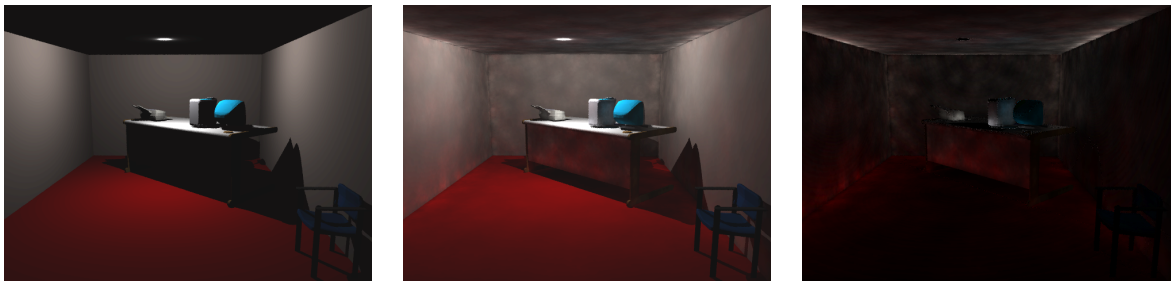
c. d.

Figure 6.24 – *a. Image obtenue sans filtre avec une recherche des* 100 *plus proches photons. - b. Image avec un filtre en cône - c. Image avec le filtre gaussien proposé par Jensen - d. Image obtenue à l'aide d'un filtre gaussien normalisé.*

L'intérêt de notre structure pour la simulation d'éclairage par lancer de photons est que les calculs de visibilité sont estimés par lots. Ce traitement par lots est réalisé au fur et à mesure des calculs, contrairement aux méthodes existantes [AB90, TFFH94, MB99, MBSB03]. Pour ces méthodes, le PVS complet associé à une pièce doit être chargé en mémoire impliquant un nombre de primitives géométriques important. Avec notre méthode, la taille maximale nécessaire ne dépend que de la pièce la plus détaillée, aucune autre pièce n'étant chargée en mémoire. Cette technique s'apparente plus aux travaux proposés par Arnaldi *et al.* [APRP97] pour la parallélisation de l'algorithme de radiosité, mais avec une structure de données plus complète à base topologique. Notons que notre structure topologique est suffisamment riche pour permettre d'appliquer les méthodes classiques de radiosité, puisque nous possédons déjà un découpage propre de la scène en pièces et en meubles, ainsi qu'un niveau peu détaillé (étage décomposé en pièces vides) permettant de calculer les relations de visibilité entre pièces.

Pour chaque pixel de l'image **faire**
  Créer un rayon R allant de l'observateur au centre du pixel **;**
  Calculer l'intersection la plus proche P_min entre la scène et le rayon R **;**
  // Ajout de l'ambiant à l'éclairement global
  $I_p \leftarrow Ambiant$; *// Calcul de l'illumination primaire à l'aide de rayons lumineux*
  Pour chaque source lumineuse **faire**
    Envoyer un rayon d'ombre **;**
    Si (aucune intersection) **alors**
      Ajouter la contribution de cette source à I_p **;**
    Fin Si
  Fin Pour
  // Calcul de l'illumination secondaire à l'aide de la photon map
  $I_p \leftarrow I_p +$ Estimation_Eclairement(P_min) **;**
  *// Calcul de la couleur du pixel : éclairement * BRDF*
  $Couleur \leftarrow I_p * f_r$ **;**
Fin Pour

Figure 6.25 – *Génération de l'image pour des surfaces diffuses avec un lancer de rayons et une récupération des photons.*



a. b. c.

Figure 6.26 – *a. Image obtenue avec prise en compte de l'ambiant mais sans illumination indirecte. - b. Image obtenu avec l'algorithme mixte lancer de rayons/lancer de photons. - c. La différence entre les deux images pour mettre en évidence les inter-réflexions qui ont été ajoutées.*

Comme nous l'avons vu, le point critique de cet algorithme est le chargement des pièces. Nous pensons donc que des études sur la gestion mémoire peuvent encore être réalisées à ce niveau. Lorsque nous traitons la propagation des photons dans une pièce, il serait intéressant que ses pièces adjacentes soient aussi présentes en mémoire. Ainsi lors du passage à une autre pièce restant à traiter (en général adjacente), les pièces proches résideraient déjà en mémoire et ceci permettrait un gain de temps au niveau du chargement. Pour ce faire, nous aurions besoin d'un cache en mémoire contenant plusieurs pièces. Les pièces y seraient chargées en fonction des besoins et en essayant de respecter des contraintes de proximité. Puisque nous faisons de la

Pour chaque pixel de l'image **faire**
  Soit N_e le nombre d'échantillons à calculer ;
  $Somme\_couleur \leftarrow 0$;
  Pour échantillon **de** 1 **à** N_e **faire**
    Prendre un point P aléatoire à l'intérieur du pixel ;
    // Calcul de chaque échantillon comme précédemment
    Créer un rayon R allant de l'observateur à P ;
    Calculer l'intersection la plus proche P_min entre la scène et le rayon R ;
    // Utilisation de rayons d'ombres et de la photon map
    Calculer l'illumination I_p au point P_min ;
    // Ajout de l'apport de ce rayon
    $Somme\_couleur \leftarrow I_p * f_r$;
  Fin Pour
  // Affectation de la moyenne des couleurs au pixel
  $Couleur\_Pixel \leftarrow Somme\_couleur/N_e$;
Fin Pour

Figure 6.27 – *Génération de l'image avec sur-échantillonage.*

propagation pièce après pièce, ceci permettrait d'avoir les pièces les plus souvent touchées directement en mémoire (comme les couloirs par exemple). Malheureusement, la taille des pièces est très variable. Il est donc difficile de dire quand les voisins d'une pièce peuvent loger en mémoire. Il faudrait pour cela fixer un nombre maximum de triangles à charger, et essayer de trouver des solutions de chargement des pièces avec un algorithme proche des algorithmes résolvant les problèmes dits de *sac à dos* : comment loger un maximum d'objets ayant chacun un poids différent à l'intérieur d'un sac de contenance restreinte.

Toujours afin de limiter les chargements de pièce pour un trop petit nombre de photons, nous pouvons modifier le choix de la pièce suivante. Actuellement, lorsqu'une pièce a fini d'être traitée, nous choisissons la suivante dans un ordre décidé arbitrairement au début de l'algorithme. Ce choix n'est bien sûr pas suffisamment pertinent. Il serait intéressant de modifier cet ordre en prenant toujours, par exemple, la pièce possédant le plus de photons à propager. Ainsi, le chargement des pièces ayant très peu de photons sera retardé au maximum en espérant que les pièces voisines lui en donnent plus. Ceci permettrait en outre d'avoir une meilleure gestion du seuil de photons puisque le dernier moment est toujours attendu pour charger la pièce. Pour le faire, nous pourrons nous inspirer des méthodes d'ordonnancement proposées par Teller [TFFH94] et Meneveaux [MBM98].

Remarquons aussi qu'une des explications du temps nécessaire au chargement est le temps de création de la grille régulière à chaque nouvelle pièce. Comme nous l'avons proposé pour l'algorithme de lancer de rayons, il serait donc intéressant de précalculer les grilles régulières de chaque pièce et de les stocker sur le disque sous forme de données binaires prêtes au chargement. Nous aurions alors un chargement beaucoup plus rapide puisqu'il ne nécessite pas de calculs consistant à placer les triangles dans la grille.

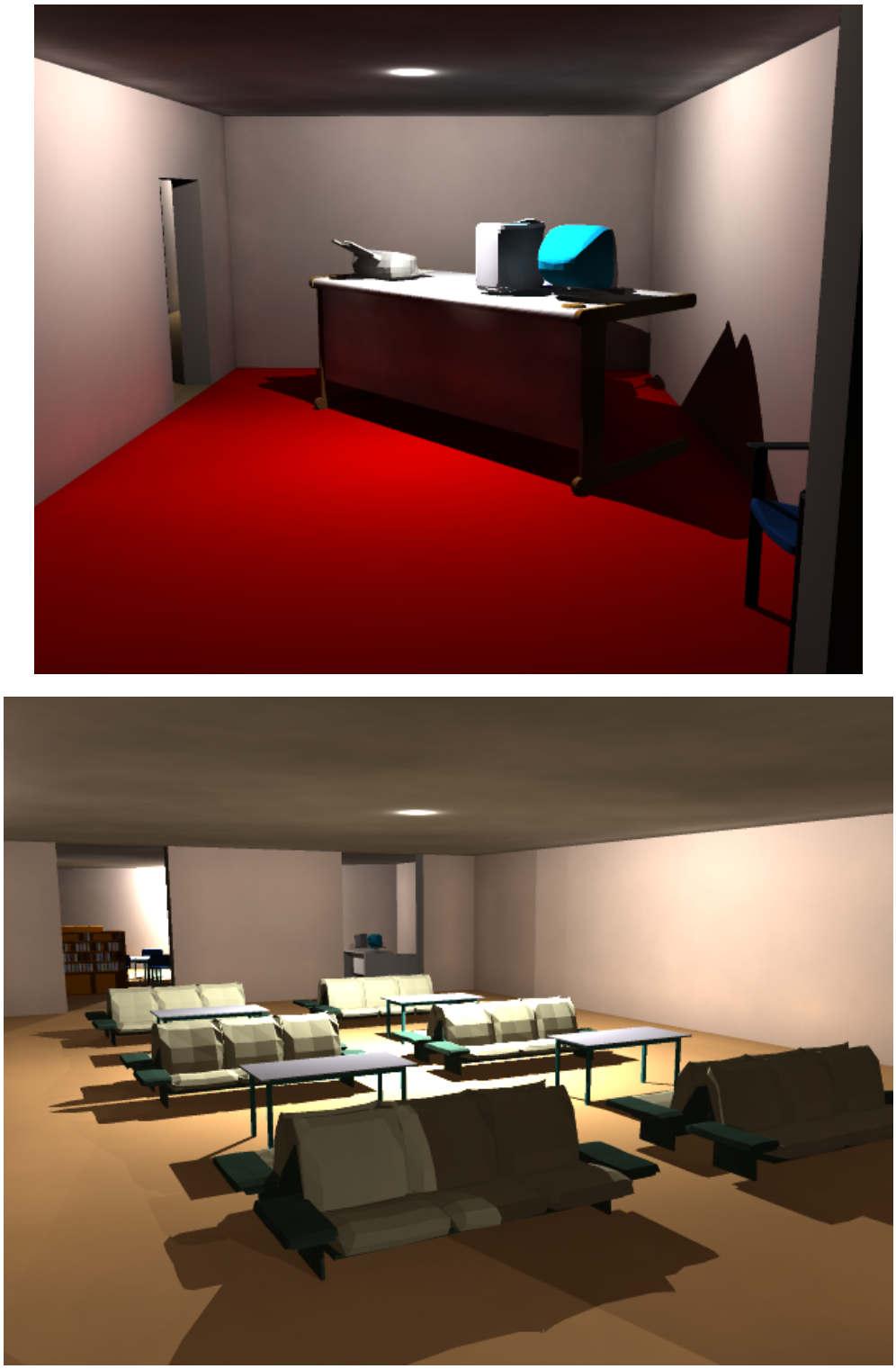


Figure 6.28 – *Images finales de nos bâtiments obtenues à l'aide d'un sur-échantillonage :* 9 *rayons par pixel.*

Pour la génération des images, nous avons vu qu'il est possible d'utiliser un sur-échantillonage pour réduire le bruit au niveau de l'image finale. Ce sur-échantillonage implique le lancer de nombreux rayons supplémentaires, ce qui augmente considérablement le temps de génération des images. Au lieu d'utiliser une carte de photons, Stürzlinger et Bastos [SB97] proposent d'utiliser des textures. Cette technique s'appelle l'écrasement de photons ou *photon splatting*. Les photons atteignant une face sont ajoutés à la texture de la face à l'aide d'une gaussienne, chacune initialisée au départ à la couleur de la face. Ceci permet d'éviter la carte de photons, la recherche des photons dans celle-ci et la multiplication du nombre de rayons à lancer. Par contre, la gestion des textures de chaque face demande une grande quantité de mémoire, ce qui dans un bâtiment devient difficile à mettre en place. Il serait donc intéressant de réfléchir à ce problème dans le cas de complexes architecturaux pour ainsi avoir une visite interactive en tampon de profondeur d'un bâtiment dont les textures représentent l'éclairement global.

Comme nous l'avons dit pour le lancer de rayons, le format de stockage des pièces sur le disque n'a pas encore été optimisé et la grille régulière devra peut-être être remplacée par une autre structure accélératrice. Ceci pourrait encore accélérer les temps de calcul. De plus, il est intéressant de remarquer que dans le cas du lancer de photons, la grille régulière sert à découper l'espace, et que l'arbre kD permet de stocker la répartition des photons dans ce même espace. Il serait donc utile de réfléchir à une structure de données permettant de stocker à la fois les faces et les photons qui les touchent. Nous aurions ainsi une meilleure connaissance de la géométrie de la scène et de la disposition de ses photons, ce qui permettrait certainement un préfiltrage des recherches de photons sur une face. Un compromis est à trouver entre l'arbre kD qui reste indépendant de la géométrie de la scène et le stockage des photons au niveau de chaque face qui est rapide, mais trop coûteux en mémoire pour un grand bâtiment.

CHAPITRE 7

CONCLUSION

La difficulté principale de la gestion des scènes architecturales complexes concerne la quantité de données à traiter. Le nombre de primitives géométriques pour un grand bâtiment meublé est tel que la manipulation en devient très difficile aussi bien du point de vue de l'édition de l'objet que du point de vue de sa visualisation. Cette grande masse de données occupe une place en mémoire supérieure aux capacités des machines classiques, et la durée des opérations de l'application devient si grande que des stratégies spécifiques doivent être mises en place.

Dans la partie consacrée à l'état de l'art, nous avons pu voir qu'en fonction du domaine de travail (modélisation ou simulation d'éclairage) les stratégies proposées diffèrent. Nous avons cherché à synthétiser l'ensemble de ces notions pour proposer une structure permettant de faire le lien entre la modélisation d'objets complexes et le rendu réaliste dans de grands bâtiments. Les différents points fondamentaux pouvant être retrouvés dans les modèles proposés ont pour but de structurer l'objet architectural :

- subdivision en pièces et ouvertures, informations sémantiques sur les objets du bâtiment ;
- informations d'adjacence (quelle ouverture donne sur telle pièce) et d'incidence (la face est partagée par deux ouvertures) ;
- hiérarchisation des données : niveaux de détails (traitement d'un niveau simplifié plus aisé que la totalité du bâtiment), inclusion (tel meuble est dans tel volume).

Le modèle à base topologique proposé étend un modèle ordonné : les cartes généralisées. Nous pouvons ainsi représenter de manière exhaustive les relations topologiques entre les différentes cellules de l'objet (sommets, arêtes, faces, volumes). Nous avons ajouté aux cartes généralisées deux notions :

- la multipartition permettant de représenter plusieurs décompositions d'un objet donné, par exemple le regroupement des pièces d'un bâtiment selon leurs sémantiques ;
- une hiérarchie de détails, souvent utilisée en visualisation, et permettant la construction d'un bâtiment par décomposition progressive.

Nous avons implanté ce modèle à partir du noyau topologique conçu et développé au laboratoire SIC. Nous avons ensuite enrichi ce code avec des opérations géométriques dédiées aux complexes architecturaux : créer les étages d'un bâtiment à partir d'un profil, construire des murs, meubler des pièces, insérer des portes et des fenêtres, etc. Ces opérations spécifiques nous

ont permis de développer un prototype de modeleur de bâtiments. Il a permis de modéliser un bâtiment d'un million de polygones et un de cinq millions de polygones.

À l'aide de l'ensemble des informations contenues dans le modèle proposé, nous avons optimisé des algorithmes classiques de visualisation et de simulation d'éclairage : le tracé de rayons, le lancer de rayons récursif et le lancer de photons. Nous avons souhaité montrer qu'une structure topologique précise permettait le déploiement de stratégies efficaces pour le calcul d'intersection rayon/scène dans le cas des grands bâtiments. Ces stratégies sont évidemment inspirées des travaux déjà existants en synthèse d'images, mais notre modèle étant beaucoup plus riche qu'une simple liste de faces, les algorithmes n'en sont que plus efficaces.

Le premier algorithme étudié pour la visualisation est le tracé de rayons. Pour réduire le nombre de tests d'intersection, nous proposons de les calculer dans la pièce où se situe l'observateur, avant d'examiner le reste du bâtiment. Le nombre restreint de polygones à traiter permet d'accélérer considérablement les calculs. Pour chaque pièce, une grille régulière est utilisée afin de calculer les intersections plus rapidement. Une fois les rayons tracés dans la pièce, nous propageons dans les pièces adjacentes ceux qui ont intersecté des ouvertures. De la même façon, un lancer de rayons récursif a été mis en place et donne des temps de calcul satisfaisants.

Nous nous sommes ensuite intéressés à un algorithme de simulation d'éclairage, le lancer de photons, en utilisant le même type de stratégies. Nous devons néanmoins gérer plusieurs dizaines de millions de photons au lieu de quelques millions de rayons. Cette augmentation brutale du nombre d'éléments à traiter nous a amenés à réfléchir plus précisément à des stratégies de gestion de la mémoire. Pour un bâtiment de 5 millions de polygones, plusieurs giga-octets de données doivent être manipulés. Les résultats que nous obtenons sont très encourageants et montrent que posséder un maximum d'informations sur un bâtiment autorise un grand choix dans les politiques de gestion.

Pour résumer, nous avons proposé un modèle de représentation d'environnements complexes d'intérieur. Cette structure hiérarchique à base topologique avec multipartitions a servi de base à un prototype de modeleur de grands bâtiments que nous avons développé. Des bâtiments de plusieurs millions de polygones ont ainsi pu être modélisés. Ce modèle nous fournit aussi un nombre important d'informations pouvant être exploitées pour réaliser des calculs de rendu réaliste efficaces et une gestion adaptée de la mémoire. Par exemple, pour une éclairement global en lancer de photons, nous obtenons des temps de calcul de quelques minutes à l'aide d'un algorithme de tracé de rayons optimisé.

7.1 Perspectives

Au cours de ces trois années de travail, nous avons privilégié certains axes de recherche afin d'obtenir des résultats concrets en visualisation d'environnements complexes. Certains points peuvent être approfondis, aussi bien dans le domaine de la modélisation qu'en visualisation et simulation d'éclairage. Il s'agit de points techniques ou de points plus théoriques, que nous exposons ici.

7.1.1 En modélisation

Dans cette section, nous précisons tout d'abord la nécessité de comparer notre modèle hiérarchique avec multipartitions à d'autres structures topologiques spécialisées. Ensuite, nous nous détaillons les informations et opérations à ajouter à notre modeleur.

La structure topologique

Actuellement, un travail a été commencé afin de prouver la cohérence de la structure hiérarchique. Des démonstrations permettent entre autres de montrer :

- que les définitions de la liaison hiérarchique et des liaisons groupantes correspondent bien à celles de la hiérarchie et des partitions multiples par étiquetage ;
- et que chaque partition d'une multipartition est une carte généralisée, i.e. les liaisons groupantes α_g sont bien des involutions au sens des cartes généralisées.

Ceci permet de montrer que le modèle que nous proposons est correctement défini. Néanmoins, comme nous l'avons vu dans l'état de l'art, d'autres structures topologiques permettent la modélisation d'objets complexes. Un travail de comparaison de notre modèle avec ces structures permettrait de mettre en évidence les avantages et les inconvénients de celui-ci, afin de l'améliorer. En effet, il a été développé essentiellement pour la modélisation de complexes architecturaux d'intérieur, c'est-à-dire des subdivisions de $\mathbb{R}^3$. Nous avons néanmoins essayé de la concevoir de façon à modéliser de manière plus générale des subdivisions hiérarchiques de $\mathbb{R}^n$. Pour atteindre ce but, un travail plus approfondi est encore nécessaire.

Exploitation pour la simulation de propagation d'ondes radioélectriques

Cette thèse a été préparée au laboratoire SIC (Signal, Image et Communication), qui est un laboratoire pluridisciplinaire. Un des axes de recherche concerne les communications sans fil. Lors du développement de notre modeleur, l'équipe travaillant sur les systèmes de transmissions nous a fait part de ses besoins en modélisation d'environnements d'intérieur réalistes. En effet, cette équipe développe actuellement un simulateur de propagation d'ondes radioélectriques au sein de bâtiments. Afin de montrer la correction de leurs calculs, ils doivent les comparer à des mesures réelles. Pour cela, le bâtiment dans lequel s'effectue la mesure doit être modélisé.

Nous avons donc développé au sein de notre modeleur un module permettant de caractériser les matériaux de construction (murs, sols, plafonds) et d'exporter le bâtiment dans un format lisible par leur programme de simulation (format MSDL). Des premiers essais tout à fait concluants ont été réalisés. La figure 7.1 montre notre modèle géométrique simplifié du bâtiment SP2MI dans lequel une simulation de propagation d'ondes a été calculée.

Ce ne sont néanmoins que des résultats embryonnaires. Une réflexion sur ce thème pourrait certainement révéler l'utilité des informations topologiques contenues dans notre modèle pour l'accélération des calculs de simulation de propagation d'ondes. En effet, ces calculs reposent sur la recherche de chemins d'ondes. Les algorithmes sont inspirés de ceux de tracé de chemins lumineux (*path tracing*) et prennent en données d'entrée des listes de faces. Nous pourrions, comme nous l'avons fait pour le lancer de rayons et le lancer de photons, réfléchir à des optimisations de ces calculs à l'aide de la topologie.

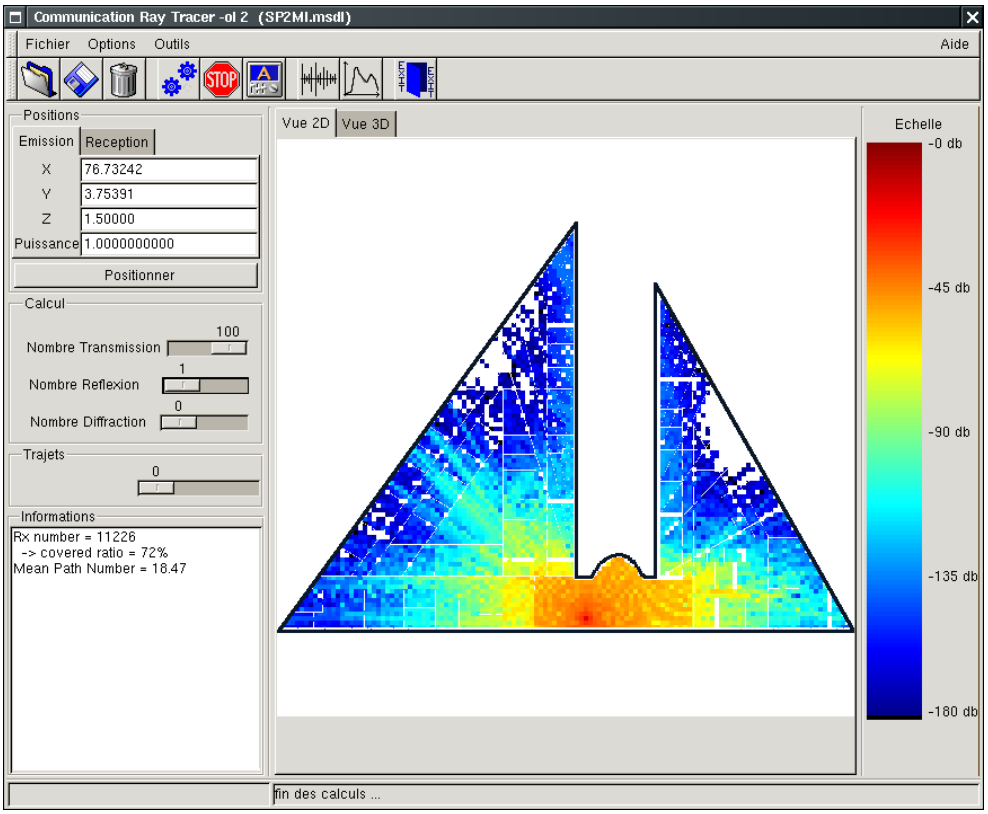


Figure 7.1 – *Simulation de propagation d'ondes dans le bâtiment SP2MI.*

Enfin, pour l'instant, les calculs ne s'opèrent que sur les murs du bâtiment en raison des mêmes difficultés rencontrées en visualisation : espace mémoire et temps de calcul. Pour cette raison, nous pensons que notre modèle peut ici être utile. Un premier stage de DESS d'électronique a été réalisé cette année dans ce but. Deux bâtiments différents ont été modélisés et les simulations ont été effectuées ; les mesures *in situ* n'ont pas encore été réalisées.

L'ameublement des pièces

Malgré les efforts réalisés pour simplifier la modélisation, l'ameublement reste fastidieux pour de très grands bâtiments. L'ensemble des fonctionnalités que nous avons développées (placement manuel des meubles, copies, scripts) reste insuffisant et l'ameublement de nombreuses pièces prend plusieurs journées de travail : il faut en effet placer les meubles un par un dans chaque pièce. C'est de cette manière qu'a été modélisé le bâtiment en forme de L (figure 4.13).

Pour simplifier cette tâche manuelle, nous avons développé des scripts d'ameublement qui permettent de répéter plusieurs fois le même ameublement dans plusieurs pièces (présenté dans le chapitre 4.4). Cette technique a été utilisée pour meubler le bâtiment octogonal (figure 4.14).

Si nous souhaitons meubler un bâtiment beaucoup plus grand de type gratte-ciel, nous devons meubler une centaine d'étages, soit plusieurs milliers de pièces. Ce travail colossal est donc diffi-

cilement envisageable manuellement. Nous avons donc commencé à approfondir notre réflexion sur l'ameublement automatique de pièces. Un stage de DESS et un projet de Licence/Maîtrise d'informatique ont été effectués afin d'évaluer les besoins et commencer le développement d'un prototype de logiciel. Le placement automatique des meubles se fait à base de règles en fonction du type de pièces souhaitées (bureau, salle de cours, etc.) et d'aléatoire pour éviter aux pièces de se ressembler. La figure 7.2 montre le résultat de l'ameublement automatique du bâtiment *Zarbi* et plus précisément une de ses pièces aux formes complexes.

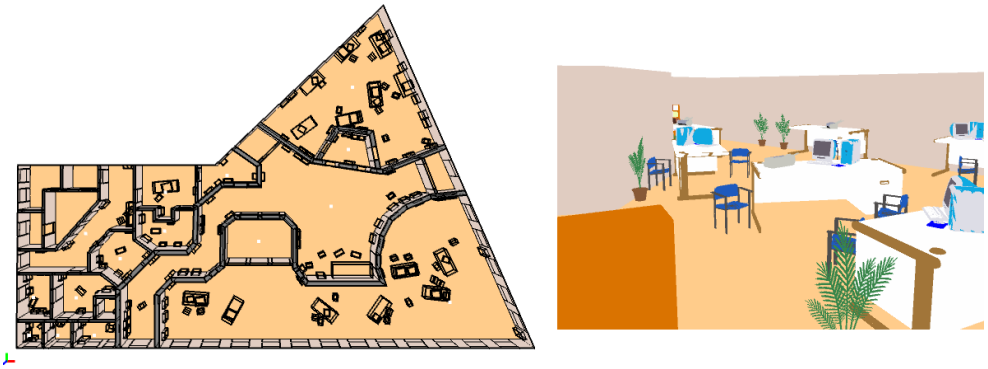


Figure 7.2 – *Images du bâtiment Zarbi meublé automatiquement (projet des étudiants de Licence/Maîtrise d'informatique, réalisé par Philippe BAUBY, Alexis DERRIEN, Damien GRAVELAT, Fabien GROSYEUX et Sébastien HORNA).*

La topologie des meubles

Nous avons noté au cours de notre travail la difficulté de retrouver des informations topologiques à partir d'une liste de faces brute. En effet, comme nous l'avons vu, la plupart des meubles utilisés n'ont pas été modélisés par nos soins. Une partie de leur topologie a été retrouvée en recherchant les sommets et les segments communs aux faces. Néanmoins ce calcul ne pouvant être complet pour les raisons précédemment citées (certaines faces manquent, modèle incohérent topologiquement, etc.), nous ne possédons pas autant d'informations sur les meubles que nous le souhaiterions : les volumes qui les constituent, les matériaux, etc. Ces informations seraient pourtant très utiles pour la visualisation et les simulations d'éclairage ou de propagation d'ondes.

Pour cette raison, nous avons souhaité créer des meubles avec leur topologie. Basé sur le noyau topologique développé au laboratoire SIC, il existe un modeleur géométrique non spécialisé nommé $Moka$. Ce modeleur (figure 7.3) offre de nombreuses opérations géométriques. Néanmoins, le nombre d'objets nécessaires à l'ameublement d'un bâtiment est très grand et les créer manuellement prendrait beaucoup de temps.

Nous avons donc aussi proposé un projet à long terme afin de créer automatiquement des éléments de mobilier avec topologie. Un premier projet de Licence/Maîtrise d'informatique a déjà eu lieu et a permis le développement de plusieurs modules de création de meubles paramétrables. Quelques résultats sont visibles sur la figure 7.4. Ces meubles sont actuellement modélisés sans

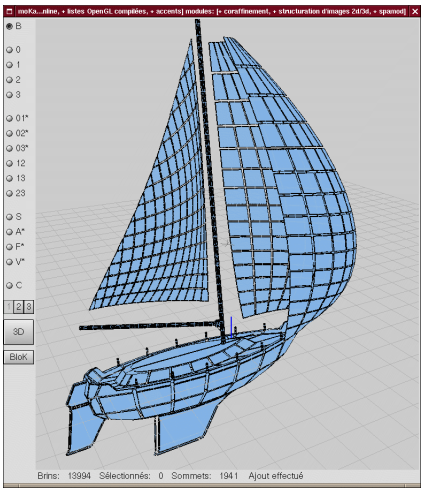


Figure 7.3 – *Image de l'interface de Moka et d'un objet entièrement réalisé avec Moka.*

informations topologiques. Nous devons donc encore améliorer ces programmes pour les rendre compatibles avec le modeleur $Moka$.

Figure 7.4 – *Meubles générés automatiquement (stage de Licence/Maîtrise d'informatique réalisé par Aurélien ANDRE, Martin DRUON, Matthieu LAGARDE et Olivier TULET).*

Le modeleur

Le modeleur de grands bâtiments n'est encore qu'un prototype. Tout d'abord, son interface n'est pas suffisamment intuitive. Cette dernière a été développée au fur et à mesure des besoins en modélisation et en visualisation. Les boutons et les menus sont classés par catégories, mais ils sont en trop grand nombre pour qu'un utilisateur puisse prendre aisément le logiciel en main. Les

boîtes de dialogue sont trop rudimentaires et il est fréquent de retrouver au niveau de l'interface des messages d'erreurs qui ne sont compréhensibles que par le développeur. Un important travail concernant l'Interface Homme Machine est donc essentiel pour que ce modeleur puisse être utilisé par d'autres personnes.

Outre ces problèmes d'interface, il est essentiel d'augmenter ses capacités de modélisation pour obtenir des bâtiments de formes plus variées. Actuellement, les étages d'un bâtiment sont obtenus par extrusion d'un même profil, ce profil étant un polygone quelconque. Il faudrait pouvoir créer des étages de profils différents, comme pour le modèle du Soda Hall (figure 1.2). De plus, si l'utilisateur souhaite créer des arrondis ou des courbes, il doit les dessiner point par point comme sur le demi-cercle du bâtiment SP2MI (figure 4.9). Il n'est pas encore possible de créer des escaliers entre les étages, les balcons n'existent pas, des liens entre étages ne sont pas prévus au niveau de l'interface (mezzanine, passerelle), etc. Bien que le noyau topologique et notre modèle hiérarchique permettent ces opérations, elles n'ont pas encore été développées. Il reste donc encore un travail important à mener sur le modeleur pour en faire un logiciel abouti capable de modéliser tout type de bâtiment.

7.1.2 En rendu réaliste

Pour obtenir des visualisations et des simulations d'éclairage plus réalistes, il faut évidemment enrichir les plongements par d'autres informations photométriques plus complètes qu'une unique couleur RVB : des textures, des BRDFs non diffuses, des sources lumineuses non ponctuelles ou des matériaux au niveau des meubles. Ceci est assez simple à réaliser puisqu'il suffit d'ajouter des classes de plongements au sein du modeleur. La difficulté sera de prendre en compte ces informations dans nos algorithmes de synthèse d'images. Dans cette section, nous nous intéressons donc surtout aux améliorations que nous pourrions apporter à ces algorithmes pour optimiser les temps de calcul, et aussi à une étude du calcul de radiosité auquel nous pensons pouvoir apporter quelques optimisations similaires à celles des lancers de rayons et de photons.

Les lancers de rayons et de photons

Dans les deux derniers chapitres de ce mémoire, nous avons expliqué en détail ces deux algorithmes et avons remarqué qu'il était encore possible de gagner en temps d'exécution en traitant mieux le chargement des pièces. En effet, actuellement les pièces sont exportées depuis le modeleur sous forme de fichier texte pour être traitées ensuite. Lors du calcul, ces fichiers sont lus et les faces sont placées à l'intérieur de la grille régulière. Ajoutés à l'analyse syntaxique du fichier texte, ces calculs nous font perdre du temps. Ceci pourrait être évité si les grilles régulières étaient pré-calculées et stockées au format binaire.

Une autre économie de temps pourrait être faite pour le calcul de l'illumination directe. Actuellement le calcul d'un rayon lumineux est très proche de celui d'un rayon primaire. Il est envoyé dans sa pièce de départ où une intersection est calculée pour savoir si le point de départ est à l'ombre. Si le rayon rencontre une face d'ouverture, le rayon est propagé vers la pièce suivante. Pour le cas particulier des scènes d'intérieur avec de nombreuses pièces, nous pouvons

remarquer que la plupart des sources lumineuses non visibles depuis un point sont cachées principalement par les murs. Il serait donc intéressant pour ces rayons de calculer en priorité les intersections avec les murs, sols et plafonds et non pas avec toute la pièce. Ainsi, nous éliminerions très rapidement toutes les sources lumineuses des autres étages et des pièces éloignées.

Enfin, nous aimerions étudier les différentes structures accélératrices (grilles, multigrilles, BSP) pour déterminer quelle est la structure la mieux adaptée au tracé de rayons dans une pièce quelconque (convexe ou non, petite ou grande, équilibrée ou non). Bien que la grille régulière soit très rapide, nous nous sommes rendu compte de son manque d'efficacité dans certaines situations. Par exemple, un long couloir en L avec des meubles ou des plantes donne une grille dont une grande partie possède des voxels vides et une autre des voxels avec de très nombreuses faces. Puisque nous proposions précédemment de précalculer les structures accélératrices des pièces pour ensuite les stocker en fichiers binaires, il serait même possible d'utiliser plusieurs structures accélératrices différentes (une pour les pièces équilibrées et une autre pour les cas particuliers) de manière à rester efficace pour toutes les configurations.

Calcul de radiosité

Au cours des travaux en visualisation et en simulation d'éclairage, nous avons essayé de dégager quelques pistes pour le calcul de radiosité. Le découpage en pièces et ouvertures est très utilisé dans ce domaine pour accélérer le calcul de visibilité entre faces. En effet, l'ensemble des informations contenues dans le modèle nous permet d'extraire pour le calcul de radiosité une structure de données identique à celle utilisée par Meneveaux [MBMD98].

Nous avons aussi remarqué, au cours du développement des lancers de rayons et de photons, que le principe de propagation des rayons n'est pas directement applicable au calcul de radiosité. En effet, les faces sont maillées et des rayons sont lancés entre chaque paire de mailles de la scène pour déterminer si elles sont visibles l'une pour l'autre. Si nous procédions de la même façon que précédemment, le nombre de rayons à stocker au niveau des ouvertures deviendrait beaucoup trop grand pour pouvoir les stocker en mémoire et les temps de calcul n'en seraient pas améliorés. Nous espérons donc travailler dans l'avenir sur cet algorithme pour proposer à l'aide de notre modèle des accélérations non pas de l'algorithme de radiosité lui-même, mais des phases de maillage et de précalcul de visibilité entre objets. Nous pensons que les informations topologiques pourront aider à définir des maillages adaptés pour la scène en s'intéressant aux liens entre les faces et aux meubles dans le voisinage, que la hiérarchie nous permettra des calculs de visibilité à différents niveaux de détails et enfin que le clonage des meubles simplifiera les calculs en ne les effectuant que sur une seule instance d'un même meuble.

BIBLIOGRAPHIE

[AB76] G. Agin and T. O. Binford. Representation and description of curved objects. *IEEE Transactions On Computers*, 25(4) :439–449, 1976. 19

[AB90] John Milligan Airey and Frederick P. Brooks, Jr. *Increasing update rates in the building walkthrough system with automatic model-space subdivision and potentially visible set calculations*. PhD thesis, The University of North Carolina at Chapel Hill, 1990. 16, 31, 93, 134

[AFF85] Silvia Ansaldi, Leila De Floriani, and Bianca Falcidieno. Geometric modeling of solid objects by using a face adjacency graph representation. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 131–139. ACM Press, 1985. 22

[Ago76] Max Agoston. *Algebraic topology : a first course*. Editions Marcel Dekker, New York, 1976. 32

[App68] Arthur Appel. Some techniques for shading machine renderings of solids. In *AFIPS 1968 Spring Joint Computer Conf.*, volume 32, pages 37–45, 1968. 10, 69

[APRP97] Bruno Arnaldi, Thierry Priol, Luc Renambot, and Xavier Pueyo. Visibility masks for solving complex radiosity computations on multiprocessors. *Parallel Comput.*, 23(7) :887–897, 1997. 134

[Bau72] Bruce Baumgart. Winged-edge polyhedron representation. In *Technical Report CS-320, Stanford University,CA*, 1972. 19, 20, 23

[Ber92] Yves Bertrand. *Spécification algébrique et réalisation d'un modeleur interactif d'objets géométriques volumiques*. PhD thesis, Université Louis-Pasteur de Strasbourg, Octobre 1992. 33, 35

[Bri89] Erik Brisson. Representing geometric structures in d dimensions : topology and order. In *Proceedings of 5th ACM Symposium of Computational Geometry, Saarbrücken, Germany*, pages 218–227, 1989. 20, 23, 32, 35

[CDM+94] Paolo Cignoni, Leila De Floriani, Claudio Montani, Enrico Puppo, and Roberto Scopigno. Multiresolution modeling and visualization of volume data based on simplicial complexes. *Symposium on Volume Visualization*, pages 19–26, 1994. 23

[CDP95] Frédéric Cazals, George Drettakis, and Claude Puech. Filtering, clustering and hierarchy construction : a new solution for ray tracing very complex environments. In *Eurographics'95*, September 1995. 11

[COCSD03] Daniel Cohen-Or, Yiorgos Chrysanthou, Claudio Silva, and Fredo Durand. A survey of visibility for walkthrough applications. *IEEE TVCG*, 2003. 14

[COFHZ98] Daniel Cohen-Or, Gadi Fibich, Dan Halperin, and Eyal Zadicario. Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes. *Computer Graphics Forum*, 17(3) :243–254, 1998. 16

[CT82] R. L. Cook and K. E. Torrance. A reflectance model for computer graphics. *ACM Transactions on Graphics*, 1(1) :7–24, January 1982. 74

[DB00] Reynald Dumont and Kadi Bouatouch. Combining hierarchical radiosity and lods. In *IASTED, Computer Graphics and Imaging*, 2000. 14

[DDP97] F. Durand, G. Drettakis, and C. Puech. The visibility skeleton : A powerful and efficient multi-purpose global visibility tool. *Proceedings of SIGGRAPH'97*, 1997. 16

[DDTP00] Frédo Durand, George Drettakis, Joëlle Thollot, and Claude Puech. Conservative visibility preprocessing using extended projections. *Proceedings of SIGGRAPH 2000*, July 2000. 15

[DF88] Leila De Floriani and Bianca Falcidieno. A hierarchical boundary model for solid object representation. *ACM Transactions on Graphics*, 7(1) :42–60, 1988. 23

[DL03] Guillaume Damiand and Pascal Lienhardt. Removal and contraction for n-dimensional generalized maps. In *Discrete Geometry for Computer Imagery*, number 2886 in Lecture Notes in Computer Science, pages 408–419, Naples, Italy, november 2003. 33, 40

[DM01] Leila De Floriani and Paola Magillo. Multiresolution modeling of three-dimensional shapes. Chapter 2 in 3D Synthetic Environment Reconstruction, M. Abdelguerfi (Ed.), Kluwer Academic Publishers, Boston, 2001. pp. 35-59. 24

[DPM97] Leila De Floriani, Enrico Puppo, and Paola Magillo. A formal approach to multiresolution hypersurface modeling. In W. Straßer, R. Klein, and R. Rau, editors. Geometric Modeling : Theory and Practice. Springer Verlag., 1997. 14, 24, 47

[Ede87] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1987. 22

[Edm60] Jack Edmonds. A combinatorial representation for polyhedral surfaces. In *Notices of Amer. Math. Soc.*, volume 7. 1960. 23

[Elt94] Hervé Elter. *Étude de structures combinatoires pour la représentation de complexes cellulaires*. PhD thesis, Université Louis Pasteur, Strasbourg, 1994. 23, 33, 40, 46

[EMB01] Carl Erikson, Dinesh Manocha, and William Baxter. Hlods for faster display of large static and dynamic environments. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 111–120. ACM Press, 2001. 69

[FKN80] Henry Fuchs, Zvi M. Kedem, and Bruce F. Naylor. On visible surface generation by a priori tree structures. In *Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, pages 124–133, 1980. 12

[FM] David Fradin and Daniel Meneveaux. Fast global illumination for large buildings. *En cours de soumission à Graphics Interface 2005*. 133

[FML] David Fradin, Daniel Meneveaux, and Pascal Lienhardt. Topology-based modeling and rendering for complex indoor scenes. *En cours de soumission à Computer Graphics Forum*. 47

[FML02] David Fradin, Daniel Meneveaux, and Pascal Lienhardt. Partition de l'espace et hiérarchie de cartes généralisées : application aux complexes architecturaux. *Journées AFIG*, pages 199–210, 2002. 47

[FST92] Thomas Funkhouser, Carlo Sequin, and Seth Teller. Management of large amounts of data in interactive building walkthroughs. In *Proceedings of the 1992 symposium on Interactive 3D graphics*, pages 11–20. ACM Press, 1992. 69

[FTI00] A. Fujimoto, T. Tanaka, and K. Iwata. Arts : Accelerated ray-tracing system. *IEEE Computer Graphics and Applications*, 6(4) :16–26, 2000. 11

[Fun96] Thomas Funkhouser. Coarse-grained parallelism for hierarchical radiosity using group iterative methods. *ACM SIGGRAPH'96 proceedings*, pages 343–352, August 1996. 105

[GKM93] Ned Greene, Michael Kass, and Gavin Miller. Hierarchical Z-buffer visibility. *Computer Graphics*, 27(Annual Conference Series) :231–238, 1993. 16

[Gla84] Andrew Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, 4(10) :15–22, 1984. 11

[GS85] Leonidas Guibas and Jorge Stolfi. Primitives for the manipulation of general subdivisions and the computation of voronoi. *ACM Trans. Graph.*, 4(2) :74–123, 1985. 32

[GS87] Jeffrey Goldsmith and John Salmon. Automatic creation of object hierarchies for ray tracing. *IEEE Comput. Graph. Appl.*, 7(5) :14–20, 1987. 13, 81

[GTGB84] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. Modelling the interaction of light between diffuse surfaces. In *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pages 212–22, July 1984. 10, 70

[Gui00] Oskar Guilbert. *Un Modèle Hiérarchique pour la Modélisation Géométrique à Base Topologique*. PhD thesis, Université Louis Pasteur de Strasbourg, Janvier 2000. 26

[HA00] Nicolas Holzschuch and Laurent Alonso. Using graphics hardware to speed-up visibility queries. *Journal of Graphics Tools*, 5(2) :33–47, 2000. 10

[HG97] P. S. Heckbert and M. Garland. Survey of polygonal surface simplification algorithms. In *Multiresolution surface modeling (SIGGRAPH '97 Course notes #25)*. ACM SIGGRAPH, 1997. 13

[HLHS03] Jean-Marc Hasenfratz, Marc Lapierre, Nicolas Holzschuch, and François Sillion. A survey of real-time soft shadows algorithms. *Computer Graphics Forum*, 22(4) :753–774, 2003. 10

[JC95] Henrik Wann Jensen and Niels Jørgen Christensen. Photon maps in bidirectional monte carlo ray tracing of complex objects. *Computers & Graphics*, 19(2) :215–224, March 1995. 10, 70

[Jen01] Henrik Wann Jensen. *Realistic Image Synthesis Using Photon Mapping*. A. K. Peters, Natick, Massachusetts, 2001. 106

[Kaj86] James T. Kajiya. The rendering equation. *Computer Graphics*, 20(4) :143–150, August 1986. 10, 75

[KK86] Timothy L. Kay and James T. Kajiya. Ray tracing complex scenes. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 269–278, 1986. 12, 89

[Kro95] Walter Kropatsch. Building irregulars pyramids by dual graph contraction. *IEEE Proceedings. Vision, Image and Signal Processing*, 142(6) :366–374, 1995. 23

[Laf96] Eric P. Lafortune. *Mathematical Models and Monte Carlo Algorithms for Physically Based Rendering*. PhD thesis, Department of Computer Science, Katholieke Universiteit, Leuven, Belgium, February 1996. 75

[Lev99] Bruno Levy. *Topologie Algorithmique : Combinatoire et Plongement*. PhD thesis, Institut National Polytechnique de Lorraine, Octobre 1999. 25, 47

[Lew94] R. Lewis. Making shaders more physically plausible. *Computer Graphics Forum*, 13(2), September 1994. 74, 75

[LG95] David Luebke and Chris Georges. Portals and mirrors : Simple, fast evaluation of potentially visible sets. *Symposium on Interactive 3D Graphics*, pages 105–106, 1995. 16

[Lie89] Pascal Lienhardt. Subdivisions of n-dimensional spaces and n-dimensional generalized maps. In *Symposium on Computational Geometry*, pages 228–236, 1989. 20, 23, 25, 32, 33

[Lie91] Pascal Lienhardt. Topological models for boundary representation : a comparison with n-dimensional generalized maps. *Computer Aided-Design*, 23(1) :59–82, 1991. 23

[Lie94] Pascal Lienhardt. N-dimensional generalized combinatorial maps and cellular quasi-manifolds. *International Journal of Computational Geometry and Applications*, 4(3) :275–324, 1994. 7, 32

[LMA+01] Franc Ledoux, Jean-Marc Mota, Agnès Arnould, Catherine Dubois, Pascale Le Gall, and Yves Bertrand. Spécification formelle du chanfreinage. In *AFADL'2001*, pages 157–171, 2001. 33

[Lue01] David P. Luebke. A developer's survey of polygonal simplification algorithms. *IEEE Computer Graphics and Applications*, 21(3) :24–35, 2001. 13

[MB99] Daniel Meneveaux and Kadi Bouatouch. Synchronisation and load balancing for parallel hierarchical radiosi ty of complex scenes on a heterogeneous computer network. *Computer Graphics Forum*, 18(4), December 1999. 105, 134

[MBM98] Daniel Meneveaux, Kadi Bouatouch, and Eric Maisel. Memory management schemes for radiosity computation in complex environments. In *Proceedings of Computer Graphics International 1998*. IEEE Computer Society, 1998. 136

[MBMD98] Daniel Meneveaux, Kadi Bouatouch, Eric Maisel, and Romuald Delmont. A new partitioning method for architectural environments. *The Journal of Visualization and Computer Animation*, 9(4) :195–213, 1998. 17, 31, 93, 146

[MBSB03] Daniel Meneveaux, Kadi Bouatouch, Gilles Subrenat, and Philippe Blasi. Efficient clustering and visibility calculation for global illumination. In *Proceedings of AFRIGRAPH 2003*, pages 87–94. ACM Press, 2003. 16, 17, 69, 105, 134

[Men98] Daniel Meneveaux. *Simulation D'éclairage dans des environnements architecturaux complexes : sequentielle et parallele*. PhD thesis, Institut National de Recherche en Informatique (IRISA), Rennes, France, June 1998. 71

[MPB03] Jean-Eudes Marvie, Julien Perret, and Kadi Bouatouch. Remote interactive walkthrough of city models. In *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, page 389. IEEE Computer Society, 2003. 14

[MS95] Paulo Maciel and Peter Shirley. Visual navigation of large environments using textured clusters. In *Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 95–105. ACM Press, 1995. 69

[NPG04] Mangesh Nijasure, Sumanta Pattanaik, and Vineet Goel. Real-time global illumination on gpu. In *Journal of Graphics Tools*, September 2004. 10

[NRH+77] Fred E. Nicodemus, J. C. Richmond, J. J. Hisa, I. W. Ginsberg, and T. Limperis. *Geometrical Considerations and Nomenclature for Reflectance*. Monograph number 160. National Bureau of Standards, 1977. 74

[PFP95] Valerio Pascucci, Vincenzo Ferrucci, and Alberto Paoluzzi. Dimension-independent convex-cell based hierarchical polyhedral complex : Representation scheme and implementation issues. In *SMA '95 : Proceedings of the Third Symposium on Solid Modeling and Applications*, pages 163–174, 1995. 23

[PH97] Jovan Popović and Hugues Hoppe. Progressive simplicial complexes. In *SIGGRAPH 97 Proc.*, pages 217–224, 1997. 14

[Pho75] Bui-T. Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6) :311–317, June 1975. 74

[PK86] F. Post and F. Klok. Deformations of sweep object in solid modeling. *Proc. Eurographics'86*, pages 103–113, Août 1986. 19

[RV82] A. A. G. Requicha and H. B. Voelcker. Solid modeling : a historical summary and contemporary assessment. *IEEE Comput Graphics Appl*, 2(2) :9–24, March 1982. 19

[SB97] Wolfgang Stürzlinger and Rui Bastos. Interactive rendering of globally illuminated glossy scenes. In Julie Dorsey and Philipp Slusallek, editors, *Eurographics Rendering Workshop 1997*, pages 93–102, New York City, NY, 1997. Springer Wien. 138

[SDDS00] Gernot Schaufler, Julie Dorsey, Xavier Décoret, and François Sillion. Conservative volumetric visibility with occluder fusion. In Kurt Akeley, editor, *Computer Graphics Proceedings*, Annual Conference Series, pages 229–238. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000. 15

[SGG+00] Pedro V. Sander, Xianfeng Gu, Steven J. Gortler, Hugues Hoppe, and John Snyder. Silhouette clipping. *Proceedings of SIGGRAPH 2000*, July 2000. Held in New Orleans, Louisianna. 13

[Ska01] Xavier Skapin. *Utilisation du produit cartésien en modélisation géométrique 4D pour l'animation*. PhD thesis, Université de Poitiers, 2001. 33

[Sob89] Cathy Sobhanpanah. Extension of a boundary representation technique for the description of n dimensional polytopes. *Computer & Graphics*, 13(1) :17–23, 1989. 22

[TFFH94] Seth Teller, Celeste Fowler, Thomas Funkhouser, and Pat Hanrahan. Partitioning and ordering large radiosity computations. In *Computer Graphics Proceedings, Annual Conference Series*, pages 443–450, 1994. 69, 105, 134, 136

[TS91] Seth J. Teller and Carlo H. Séquin. Visibility preprocessing for interactive walkthroughs. *Computer Graphics*, 25(4) :61–68, 1991. 16, 31, 69, 93

[Wal04] Ingo Wald. *Realtime Ray Tracing and Interactive Global Illumination*. PhD thesis, Saarland University, 2004. 12

[War92] G. Ward. Measuring and modeling anisotropic reflection. In ACM, editor, *Proceedings of SIGGRAPH'92*. ACM, July 1992. 74

[Wei85] Kevin Weiler. Edge-based data structures for solid modeling in curved-surface environments. *IEEE Computer Graphics and Applications*, 5(1) :21–40, Janvier 1985. 20, 23

[Wei88] Kevin Weiler. Boundary graph operators for non-manifold geometric modeling representations. In *Geometric Modeling for CAD Applications*, pages 37–66, North-Holland, 1988. 23

[Whi80] Turner Whitted. An improved illumination model for shaded display. *Commun. ACM*, 23(6) :343–349, 1980. 10, 70, 77

[WPS+03] Ingo Wald, Timothy J. Purcell, Joerg Schmittler, Carsten Benthin, and Philipp Slusallek. Realtime Ray Tracing and its use for Interactive Global Illumination. In *Eurographics State of the Art Reports*, 2003. 69, 70

[WWS00] Peter Wonka, Michael Wimmer, and Dieter Schmalstieg. Visibility preprocessing with occluder fusion for urban walkthroughs. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*, pages 71–82. Springer-Verlag, 2000. 14